WYDZIAL ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

Akademii Górniczo-Hutniczej im. St. Staszica w Krakowie

Skalowalny, komponentowy system zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych

Rozprawa Doktorska

Dominik Radziszowski

Promotor:
Prof. dr hab. inz. Krzysztof Zielinski

Kraków, listopad 2006

Serdecznie dziekuje wszystkim osobom, które wspieraly mnie w czasie prac nad ta rozprawa.

Jestem wdzieczny przede wszystkim mojemu promotorowi, Panu Profesorowi Krzysztofowi Zielinskiemu, za okazana pomoc oraz wiele cennych i konstruktywnych uwag.
Dziekuje kolegom z Zespolu Systemów Rozproszonych Katedry Informatyki AGH, a w szczególnosci Pawlowi Slowikowskiemu i Kazimierzowi Balosowi, za sugestie oraz czas poswiecony na dlugie dyskusje. Szczególne podziekowania kieruje do mojej zony Moniki, za wyrozumialosc i okazane wsparcie.

Dominik Radziszowski

Spis Tresci

Spis Schematów i Ilustracji

Spis tabel

1 Wstep

Wspólczesne systemy informatyczne generuja coraz wieksze ilosci informacji dotyczacych swojego dzialania, przebiegu kontrolowanych przez siebie procesów oraz zdarzen zachodzacych w otaczajacej je rzeczywistosci. Obszar dzialania systemów informatycznych powieksza sie nieustannie, obejmujac zastosowania, które jeszcze do niedawna zdawaly sie byc pod wylaczna kontrola czlowieka.

Inteligentne radarowe systemy kontroli predkosci w samochodach, zapewniajace utrzymanie bezpiecznej odleglosci od poprzedzajacego pojazdu [Hor05], samoobslugowe sklepy, w których dzieki technologii RFID nie ma kasjerek, a towary na pólkach sygnalizuja uplyniecie terminu przydatnosci do spozycia [IBM05], systemy wspierajace lekarzy, policjantów, strazaków.... Wszystkie te systemy maja zdolnosc do komunikowania sie; sa okreslane mianem „networked", "Internet ready" czy „e-business compliant". Systemy **rozproszone** (ang. distributed), bo wlasnie do takiej klasy naleza wszystkie te systemy, otaczaja wspólczesnego czlowieka coraz gesciej, i jest on od nich coraz bardziej uzalezniony. Systemy te generuja dane, olbrzymie ilosci danych. Czesc z tych danych jest przetwarzana, czesc jest agregowana i sluzy opracowaniom róznych statystyk, pozostale to 'czarna skrzynka' systemu – sa zapisywane jedynie „na wszelki wypadek" i prawdopodobnie nigdy nie beda wykorzystane. Szczególowe informacje na temat rozmów telefonicznych, gromadzone przez operatorów telefonii bezprzewodowej, zawieraja nie tylko dane o numerze i czasie polaczenia, które mozna znalezc na billingu, ale równiez dane dotyczace lokalizacji rozmawiajacych w momencie rozmowy. Komisja Europejska rozwaza zobowiazanie dostawców uslug internetowych (ang. Internet Service Providers) do przechowywania danych na temat odbieranych i wysylanych listów elektronicznych [Wyb05]; czujniki dymu, wilgotnosci, ruchu, zapylenia, swiatla w coraz bardziej inteligentnych budynkach – dane, gigantyczne ilosci danych, pochodzacych z monitorowania systemów rozproszonych, które musza byc zbierane i przechowywane.

Sytuacja ta stanowi wyzwanie dla wspólczesnych systemów zbierajacych i przechowujacych w bazie danych dane pochodzace z monitorowania, glównie w dwóch plaszczyznach: uniwersalnosci oraz skalowalnosci. Uniwersalnosc systemów polega, w tym kontekscie, na zdolnosci do przechowywania dowolnych, podlegajacych monitorowaniu danych, adaptowalnosci do ich zmiennosci oraz wsparcia dla róznych trybów monitorowania.

Skalowalnosc ma sluzyc zapewnieniu odpowiedniej wydajnosci, pomimo wzrostu wielkosci strumienia danych przyjmowanych przez system oraz objetosci danych juz zapisanych.

Systemy komponentowe

Fakt dynamicznego rozwoju technologii komponentowych oraz trend w zakresie ich powszechnego wykorzystywania, stawia systemy komponentowe w centrum zainteresowania oraz czyni badania nad ich skalowalnoscia i wydajnoscia szczególnie aktualnymi. Prowadzone w tym obszarze prace skupiaja sie glównie na tworzeniu rozwiazan ogólnych, wlasciwych dla dowolnych obiektowych struktur danych, optymalizowanych pod katem operacji odczytu poprzez wprowadzanie wielopoziomowych mechanizmów pamieci podrecznej. Stosunkowo malo jest prac dotyczacych porównania i optymalizacji architektur aplikacji komponentowych pod katem wydajnosci uzyskiwanej w procesie zapisu duzej ilosci danych. Powyzsze uwarunkowania spowodowaly wybranie technologii komponentowej jako bardzo atrakcyjnego obszaru dla realizacji systemu zbierania i przechowywania danych, który moze sprostac wymaganiom, zarówno w zakresie ogólnosci rozwiazania jak i jego skalowalnosci.

> **Komponent** – rozszerza koncepcje programowania obiektowego, jest konfigurowalnym zbiorem obiektów, posiadajacym dobrze zdefiniowana, wyspecjalizowana logike dzialania. Jest on uruchamiany i dziala w ramach kontenera komponentów, poprzez który udostepnia interfejsy realizujace okreslona funkcjonalnosc.
>
> **Kontener komponentów** – jest srodowiskiem dzialania komponentów, którym udostepnia szereg, niezbednych do ich prawidlowego dzialania uslug. Uslugi te dzialaja w wiekszosci w ramach **serwera aplikacji**, którego kontener jest czescia, obejmuja one miedzy innymi mechanizmy autoryzacji, uwierzytelniania, równowazenia obciazenia oraz zarzadzanie cyklem zycia komponentów.

Podejscie komponentowe wymusza taka budowa systemu, w której jego poszczególne elementy sa od siebie w duzym stopniu niezalezne, a komunikacja zachodzi jedynie przez dobrze zdefiniowane interfejsy. System moze byc tworzony przede wszystkim z uwzglednieniem wymagan funkcjonalnych. Po zaimplementowaniu niezbednej funkcjonalnosci, jest on weryfikowany i badany pod wzgledem wydajnosciowym. Komponentowa budowa umozliwia niezalezne strojone poszczególnych elementów (komponentów) systemu oraz ich zastepowanie bardziej wydajnymi odpowiednikami. Uprasza równiez, uzyskanie wlasnosci skalowalnosci systemu oraz zrównoleglenie

przetwarzania poprzez zwielokrotnienie ilosci instancji komponentów dzialajacych w ramach jednego badz kilku kontenerów komponentów.

> **Skalowalnosc systemu komputerowego** – zdolnosc systemu do utrzymania parametrów jakosciowych (ang. QoS - Quality of Service) pomimo zwiekszania przetwarzanego strumienia danych, uzyskiwana poprzez rozszerzanie zasobów w oparciu, o które system dziala.
>
> **Skalowalnosc wertykalna** – polega na skalowaniu systemu poprzez dodawanie dodatkowych procesorów i pamieci w ramach jednego serwera [AB03].
>
> **Skalowalnosc horyzontalna** – polega na skalowaniu systemu poprzez zwiekszanie ilosci serwerów, a wiec poszerzaniu bazy sprzetowej, na jakiej dziala system [AB03].

O ile skalowac wertykalnie mozna praktycznie dowolny system informatyczny, o tyle zdolnosc do skalowalnosci horyzontalnej jest silne zalezna od przyjetej architektury systemu oraz sposobu jego napisania. Mówiac o systemie skalowalnym, autor ma na mysli rozwiazania, które, przy skalowaniu horyzontalnym, nie wymagaja zmian w swojej architekturze.

1.1 Cel i teza rozprawy

Polaczenie mozliwosci systemów komponentowych oraz problemów zapisu duzej ilosci danych pochodzacych z monitorowania stanowi ciekawy problem badawczy. Idea ta nie zostala dotad szerzej podjeta. Producenci systemów monitorujacych koncentruja sie na tworzeniu rozwiazan zamknietych, wspólpracujacych z wlasnymi rozwiazaniami w dziedzinie monitorowania, czesto integrowanymi bezposrednio z bazami danych. Twórcy srodowisk komponentowych prowadza prace pod katem tworzenia rozwiazan ogólnych.

Autor stawia sobie za cel zbadanie, w jaki sposób, przy zachowaniu uniwersalnosci systemu zbierania i przechowywania danych pochodzacych z monitorowania, mozna zapewnic jego wydajnosc i skalowalnosc. W tym kontekscie zaproponowanych i zaimplementowanych zostalo kilka aplikacji komponentowych, opartych na wielu instancjach baz danych, mechanizmach buforowania i partycjonowania danych oraz komunikacji asynchronicznej; ich wplyw na wydajnosc i skalowalnosc zostal zweryfikowany doswiadczalnie.

Celem pracy jest wykazanie, ze w technologiach komponentowych mozliwe jest stworzenie wydajnego i skalowalnego systemu zbierania i przechowywania danych pochodzacych z monitorowania reprezentowanych obiektowo zasobów oraz takiej reprezentacji tych danych, która zapewni elastycznosc ich przechowywania i wyszukiwania.

Powyzsze rozwazania doprowadzily do sformulowania nastepujacej tezy:

> ***Mozliwa jest konstrukcja komponentowego systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych, którego dzialanie jest adaptowalne do zmian zarówno rodzaju danych pochodzacych z konkretnego zasobu jak i trybu ich zbierania, zapewniajacego odpowiednia wydajnosc i skalowalnosc.***

Zasób jest rozumiany jako dowolne urzadzenie badz aplikacja udostepniajaca interfejs programistyczny, umozliwiajacy odczyt parametrów okreslajacych jego stan. Reprezentacja obiektowa jest wymaganiem umozliwiajacym ograniczenie badan do modelu obiektowego, w którym dziala wiekszosc wspólczesnych systemów monitorujacych. Nie wplywa ono na ogólnosc rozwiazania, poniewaz inne modele danych mozna zazwyczaj opisac modelem obiektowym. **Adaptowalnosc do zmian** oznacza dzialanie systemu w warunkach duzej zmiennosci monitorowanych zasobów. Monitorowany zasób moze bowiem w dowolnym momencie zmieniac (dodawac, usuwac) parametry, jakie udostepnia do monitorowania. Wartosci róznych parametrów moga byc udostepniane przez zasób w jednym z **trybów**: raportowanie (ang. push), odpytywanie (ang. pull) oraz sledzenie zmian wartosci (ang. tracing); oraz moga byc zapisywane z rózna czestotliwoscia. Zaklada sie opracowanie uniwersalnych interfejsów programistycznych, pozwalajacych na jednolity zapis oraz odczyt danych, pochodzacych z dowolnego z monitorowanych zasobów. **Komponentowa** budowa ma na celu wykazanie, ze srodowiska komponentowe nadaja sie do implementacji tego typu systemów oraz zapewniaja odpowiednia **wydajnosc i skalowalnosc**. Konstrukcja systemu w technologii komponentowej jest równiez okazja do pokazania zalet oraz ograniczen tego podejscia.

Wykazanie prawdziwosci postawionej tezy zostalo dokonane poprzez opracowanie modelu sytemu, spelniajacego postawione postulaty oraz wykazanie jego realizowalnosci poprzez implementacje. W celu osiagniecia odpowiedniej wydajnosci i skalowalnosci stworzony model zostal poddany szeregu optymalizacjom. Calosc prac uzupelniaja testy wydajnosciowe, porównujace dyskutowane rozwiazania oraz obrazujace wplyw specyficznych, istotnych dla podejscia komponentowego rozwiazan architektury systemu na uzyskiwane wyniki.

1.2 Osiagniete rezultaty

W celu zaprezentowania problematyki rozprawy oraz wykazania prawdziwosci postawionej tezy, przedstawione zostaly wybrane zagadnienia z zakresu monitorowania systemów rozproszonych, technologii komponentowych oraz mechanizmów zwiekszania

wydajnosci systemów komponentowych. Na ich podstawie autor okreslil szczególowa liste wymagan dla SZiPD (Systemu Zbierania i Przechowywania Danych). Kolejno dyskutowane sa mozliwe rozwiazania ogólnej architektury systemu, sposoby wspólpracy z agentami istniejacych systemów monitorujacych oraz mechanizmy zbierania i udostepniania danych. Zaproponowany i szczególowo opisany zostal model informacyjny dla systemów monitorujacych oraz stworzony na jego podstawie ogólny obiektowy model danych oparty na koncepcji meta-danych; zdefiniowane uniwersalne interfejsy dostepu do danych. W celu wykazania realizowalnosci przyjetych wymagan, autor zbudowal w technologii komponentowej bazowy model systemu oraz pozytywnie zweryfikowal uzyskana przez model funkcjonalnosc.

Dalsze, opisane w niniejszej dysertacji prace mialy na celu zapewnienie odpowiedniej wydajnosci i skalowalnosci systemu. Autor badal wplyw, jaki na te parametry maja zmiany wewnetrznej architektury systemu. Przedstawione i zaimplementowane zostalo piec róznych modeli SZiPD wykorzystujacych rózne mechanizmy optymalizacji systemów komponentowych: klasteryzacje serwerów aplikacji, zwielokrotnienie instancji baz danych, partycjonowanie danych oraz komunikacje asynchroniczna i kolejkowanie z wykorzystaniem brokera komunikatów. Zaprezentowana zostala równiez autorska koncepcja modelu hybrydowego, który posiada zdolnosc adaptacji mechanizmów zapisu danych do wielkosci strumienia danych.

Przedstawione modele zostaly uruchomione na dedykowanej infrastrukturze sprzetowej oraz dokonano porównania ich wlasnosci. Autor zaproponowal metodologie testów, kryteria ewaluacji oraz srodowisko uruchomieniowe, w ramach którego przeprowadzono eksperymenty umozliwiajace praktyczna weryfikacje przedstawionych modeli w kontekscie ich wydajnosci oraz skalowalnosci.

Autor wykazal przydatnosc architektury komponentowej do tworzenia skalowalnych systemów zbierania i przechowywania duzej ilosc danych pochodzacych z monitorowania systemów rozproszonych, a tym samym wykazal prawdziwosc postawionej tezy.

1.3 Struktura pracy

Niniejsza praca posiada nastepuja strukture: Rozdzial 2 przedstawia aktualne i popularne platformy dla tworzenia systemów komponentowych wraz z analiza ich mozliwosci i ograniczen. Szerzej omówiona zostala technologia J2EE, z uwzglednieniem zagadnien zwiazanych z optymalizacja dzialania systemów komponentowych w niej

zbudowanych. W rozdziale 3 rozwazana jest ogólna architektura systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych, zaproponowany zostal model informacyjny i ogólny obiektowy model danych dla takiego systemu oraz zdefiniowane uniwersalne interfejsy dla zapisu i odczytu danych. Rozdzial konczy projekt komponentowego modelu systemu, posiadajacy postulowane wlasnosci funkcjonalne. Rozdzial 4 poswiecony jest optymalizacji modelu bazowego oraz modelom alternatywnym. Kolejne modyfikacje maja zapewnic niefunkcjonalne wlasnosci systemu obejmujace odpowiednia wydajnosc i skalowalnosc. Rozdzial 5 przedstawia metodologie badan wydajnosciowych systemu oraz zbiera i podsumowuje wyniki uzyskane z przeprowadzonych testów. Ostatnim rozdzialem pracy jest rozdzial 6 zawierajacy wnioski z pracy oraz propozycje dalszego rozwoju systemu.

2 Przeglad technologii lezacych u podstaw pracy

"Jezeli ci sie zdaje, ze wiele rzeczy
znasz i dobrze je rozumiesz,
pamietaj, ze nieporównywalnie wiecej
jest takich, których nie pojales"
- Tomasz a'Kempis

Niniejszy rozdzial zawiera koncepcje i rozwiazania z zakresu technologii komponentowych, lezace u podstaw pracy. Przyblizone zostaly rozwiazania z zakresu monitorowania srodowisk rozproszonych, które sa zródlem danych dla tworzonego systemu zbierania i przechowywania danych. Szerzej opisane zostaly: SNMP, JMX, CIM, mozliwosc monitorowania w systemach rodziny MS Windows oraz wybrane modele informacyjne. Rozdzial zawiera równiez opis wspólczesnych platform do budowy systemów komponentowych, sposród których szerzej omówiono J2EE. Przedstawiono koncepcje warstwowej budowy aplikacji oraz wlasnosci, dostepnych w technologii J2EE typów komponentów. W kolejnym punkcie opisane zostaly zagadnienia zwiekszania wydajnosci aplikacji komponentowych. Przyblizono mozliwosci w zakresie strojenia aplikacji, klasteryzacji serwerów aplikacji, wykorzystania wielu instancji baz danych, partycjonowania danych oraz uzycia brokera komunikatów. Rozdzial zakonczony jest podsumowaniem.

2.1 Wybrane rozwiazania z zakresu monitorowania systemów rozproszonych

Poruszane w rozdziale 1 zagadnienia zbierania danych z systemów rozproszonych dotycza bardzo szerokiej gamy systemów. W celu zapewnienia ogólnosci tworzonego systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych, dokonano przegladu wybranych klas systemów monitorujacych. Analizie podlegaly wykorzystywane modele informacyjne i modele danych, modele dystrybucji danych oraz ogólne architektury systemów. Przeglad obejmowal nastepujace, rózne technologicznie implementacje srodowisk przetwarzania rozproszonego:

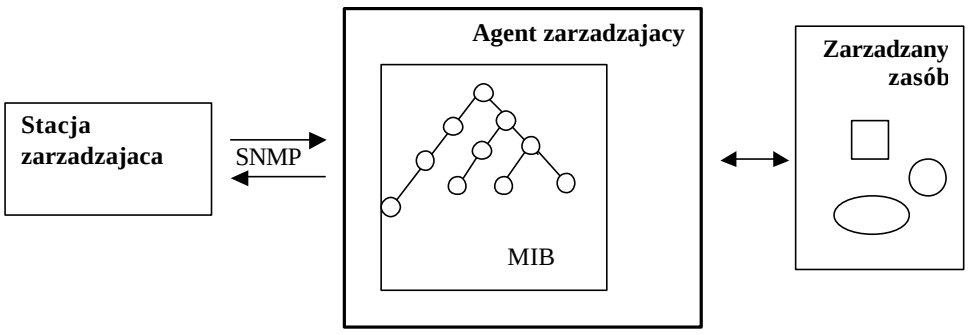
- monitorowania infrastruktury (sieciowej, serwerów, baz danych) w oparciu o standardy SNMP [CDS+88] i CIM [DMTF99][DMTF00],
- monitorowania obliczen w srodowiskach PVM i MPI [HMR95] [KG96][WL96][MR02] [DHK+94] [Fun00],

- monitorowania architektur typu GRID, autorzy [GWB+04][BKP+00] wyliczaja i opisuja ponad 25 róznych systemów monitorujacych te architekture, a problemy ich optymalizacji sa dyskutowane w [RML03] [ESK+97] [TSL+00] [LWS+03],
- monitorowania oprogramowania opartego na srodowisku CORBA [OH97] [MM97] [Sie01] [Iona99] [LRG00] [Lau95] [ZLS+95],
- monitorowania systemów wielkiej skali (ang. enterprise systems) tworzonych w technologiach dotNET i Java [ACM01] [Mar02] [KB02],
- zagadnien zwiazanych z instrumentacja zasobów podlegajacych monitorowaniu [SYS95] [Lau01] [SUN05],
- zagadnien zwiazanych z prezentacja danych pochodzacych z monitorowania [KS91] [HE98]

W celu utrzymania rozsadnej objetosci pracy, w niniejszym rozdziale przedstawiono jedynie wybrane rozwiazania. Przyblizone zostaly: protokól SNMP, monitorowanie systemów z rodziny MS Windows, specyfikacja JMX oraz model informacyjny CIM, WBEM i DEN-ng.

2.1.1 Simple Network Management Protocol

Pierwsza wersja ‘prostego protokolu zarzadzania siecia’ (SNMP - ang. Simple Network Management Protocol), zostala opisana w RFC 1067 juz w roku 1988 i byla zaprojektowana z mysla o zdalnej inspekcji sprzetowych elementów sieci TCP/IP [Sta93][CDS+88]. Wprowadzony jako rozwiazanie przejsciowe w oczekiwaniu na miedzynarodowy, zatwierdzony przez ISO standard, SNMP zyskal niespodziewanie duza popularnosc. Bazuje on na prostej architekturze do zarzadzania opartej, na elementach, których wzajemne powiazania przedstawia Rys. 1.



Rys. 1 Architektura zarzadzania SNMP.

Rola poszczególnych elementów jest nastepujaca:

Stacja zarzadzajaca (ang. management station) – jest to najczesciej komputer wyposazony w oprogramowanie kontrolujace proces zarzadzania, przetwarzajace i prezentujace monitorowane dane oraz bedaca interfejsem pomiedzy czlowiekiem (zarzadca systemu), a zarzadzanym systemem. Moze ona wysylac zadania pobrania danych oraz wykonania operacji do agenta zarzadzajacego oraz otrzymywac odpowiedzi lub inne, nie zamówione wiadomosci od agenta.

Agent zarzadzajacy (ang. management agent) – jest programem, dzialajacym na elemencie sprzetowym podlegajacym zarzadzaniu (np. komputerze, switchu, routerze), bedacym bezposrednio za nie odpowiedzialnym w procesie zarzadzania. Jego rola jest utrzymanie lokalnej informacji oraz odpowiadanie na zadania udostepnienia informacji oraz wykonania operacji, pochodzace ze stacji zarzadzajacej oraz asynchroniczne przekazywanie waznych informacji, takich jak bledne dzialanie czy uszkodzenie elementu.

Baza informacji zarzadzajacej (ang. MIB - Management Information Base) – jest drzewiasta struktura zlozona z zarzadzanych obiektów (ang. management objects), reprezentujacych docelowe obiekty wewnatrz agenta. Obiekty SNMP nie sa obiektami w rozumieniu programowania obiektowego, sa zmiennymi o typie prostym, reprezentujacymi rózne aspekty zarzadzanego zasobu, np.: ilosc pakietów transmitowanych przez interfejs routera czy przepustowosc tego interfejsu. Obiekty te sa standaryzowane dla konkretnego systemu sprzetowego (np. istnieje definicja dla huba, bridga, routera itp.). Jezykiem opisu dla struktur MIB jest ASN1 - abstrakcyjna notacja skladni (ang. Abstract Syntax Notation). Kazdy z zarzadzanych obiektów jest dostepny poprzez unikalny identyfikator obiektu, który reprezentuje ten obiekt jako lisc w globalnym drzewie nazw, zarzadzanym przez organizacje standaryzujace ISO i ITU-T. Kazdy z identyfikatorów obiektów sklada sie z rozdzielonej kropkami sekwencji liczb lub napisów (w zaleznosci od przyjetej notacji) i opisuje sciezke od korzenia drzewa nazw az do liscia reprezentujacego konkretny, szukany obiekt [Sta93] [Lau01].

Wymiana danych pomiedzy agentem a stacja zarzadzajaca realizowana jest z wykorzystaniem protokolu zarzadzania siecia (ang. Network-Management Protocol). Wiadomosci sa transmitowane w jednostkach danych protokolu (ang. PDU – Protocol Data Unit), z wykorzystaniem bezpolaczeniowego protokolu transportowego UDP. SNMP przewiduje piec wiadomosci, które funkcjonalnie zaliczaja sie do czterech grup:

- wiadomosci typu **pobierz (ang. Get)** – umozliwiaja stacji zarzadzajacej zglaszanie zadania pobrania wartosci z zarzadzanego obiektu (obejmuja wiadomosci *GetRequest* i *GetNextRequest*),
- wiadomosci typu **ustaw (ang. Set**) – umozliwia stacji zarzadzajacej ustawianie wartosci zarzadzanego obiektu (wiadomosc *SetRequest*),
- wiadomosc **odpowiedz (ang. Response) –** jest wysylana przez agenta w celu potwierdzenia otrzymania wiadomosci Get lub Set,
- wiadomosc **Trap –** umozliwia asynchroniczne notyfikowanie agenta o znaczacych zdarzeniach zachodzacych po stronie agenta.

SNMP jest uznanym i bardzo rozpowszechnionym protokolem, implementowanym w praktycznie kazdym zarzadzalnym urzadzeniu sieciowym. Fakt ten musi byc uwzgledniony przy tworzeniu system zbierania i przechowywania danych, pochodzacych z monitorowania systemów rozproszonych, tak aby pochodzace z SNMP dane mogly byc w systemie zapisywane. Przy pomocy zaproponowanego ogólnego obiektowego modelu danych – punkt 3.3 - bedzie mozna opisac dane zgodne z MIB, a komunikacja z agentami SNMP bedzie mogla byc zrealizowana z wykorzystaniem technologii JMX, opisanej w dalszej czesci rozdzialu.

2.1.2 Monitorowanie w srodowisku MS Windows

Podstawowym interfejsem do monitorowania, udostepnianym przez system Windows jest tzw. interfejs rejestru (ang. registry interface). Jest on skomplikowany i niewygodny w uzyciu. Dlatego tez nowsze systemy Microsoft serii Windows 2000/NT/XP zostaly wyposazone w duzo bardziej przejrzysty i uzyteczny z punktu widzenia programisty interfejs o nazwie PDH (ang. Performance Data Helper). PDH jest zbudowany w oparciu o interfejs rejestru i nie wprowadza zadnej nowej funkcjonalnosci, udostepnia natomiast szereg funkcji oraz struktur danych, które w znaczacy sposób upraszczaja monitorowanie [Den02].

PDH reprezentuje przeznaczone do monitorowania zasoby w postaci obiektów (ang. performance objects), oraz instancji obiektów (ang. performance object instance). Istnieje zatem np. obiekt Procesor reprezentujacy procesor na danej maszynie, oraz obiekt reprezentujacy proces, o tylu instancjach, ile jest aktualnie uruchomionych procesów. Kazdy taki obiekt posiada liste typowych dla siebie liczników (ang. performance counters). Sa to konkretne, mierzalne parametry danego zasobu, które udostepnia do monitorowania. Dla

obiektu ‘Procesor’ dostepne sa miedzy innymi takie liczniki jak np.: czas procesora, czas uzytkownika, czas uprzywilejowany. Odczyt wartosci danego licznika jest mozliwy na podstawie pelnej nazwy licznika, sklada sie ona z nazwy maszyny (mozna ja opuscic, gdy chodzi o lokalna maszyne), nazwy obiektu, nazwy instancji oraz nazwy samego licznika. Przykladowo, nazwa licznika reprezentujacego czas procesora na lokalnej maszynie ma postac: *\Processor(0)\% Processor Time*. Sporym problem przy uzywaniu nazw jest fakt, ze sa one rózne dla róznych wersji jezykowych systemu Windows. Podany przyklad jest poprawny dla angielskiej wersji systemu, podczas gdy dla polskiej wersji jezykowej nalezaloby napisac *\Procesor(0)\% Czas Procesora*.

Dostep do wartosci liczników jest mozliwy poprzez obiekty zapytania (ang. query). Obiekty te sa kolekcjami liczników, którymi zarzadza programista. Po stworzeniu takiej kolekcji mozna, za pomoca odpowiednich funkcji, uaktualniac wartosci przechowywanych w niej liczników. Dostep do wartosci bazuje na ich pobieraniu/aktualizacji poprzez wywolania odpowiednich funkcji, brak mechanizmów notyfikacji [Den02] [Cha04].

Poniewaz u podstaw pracy lezy monitorowanie systemów rozproszonych, a te coraz czesciej zawieraja elementy oparte na systemie operacyjnym Windows, przedstawiony w punkcie 3.3 ogólny model danych oraz uniwersalne interfejsy dostepu zostaly stworzone w sposób umozliwiajacy integracje z przedstawionymi powyzej mechanizmami monitorowania systemu MS Windows.

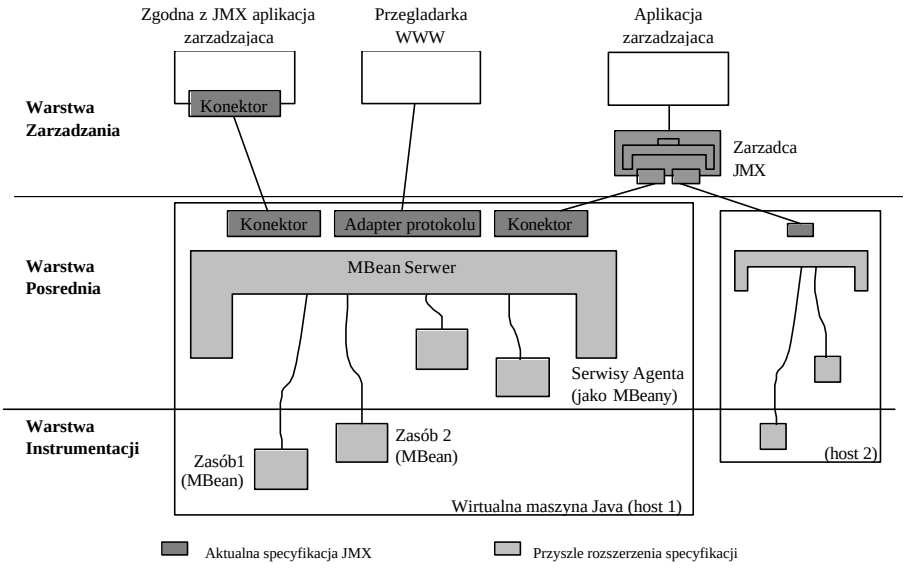
2.1.3 Java Management eXtension

Java Management Extenstions (JMX) jest kompletna specyfikacja srodowiska do zarzadzania zasobami dla jezyka Java. W sklad JMX wchodzi opis architektury, wzorców projektowych, interfejsy oraz API programistyczne. Specyfikacja JMX jest opisana w dokumentach JSR-003 [SUN04d] i JSR-160 [SUN03b], stworzonych z inicjatywy Sun Microsystems [ZL02][JK04]. Architektura JMX zostala przedstawiona na Rys. 2, wyróznia ona trzy podstawowe warstwy:

- warstwa instrumentacji (ang. instrumentation level) – odpowiada bezposrednio za komunikacje z zarzadzanym zasobem, który moze byc serwisem, urzadzeniem, aplikacja lub dowolnym innym obiektem. Jej zadaniem jest udostepnienie jednolitego interfejsu, niezaleznego od rodzaju zasobu. W JMX stanowia ja komponenty nazwane *Managed Beans (MBeans)*. MBean jest klasa Java implementujaca okreslone przez specyfikacje JMX interfejsy. Funkcjonalnosc

MBean'a obejmuje: udostepnianie informacji na temat zarzadzanego zasobu (lista metod i atrybutów udostepnionych przez zasób do monitorowania i zarzadzania), wykonywanie operacji na zarzadzanym zasobie (wywolywanie metod, ustawianie/pobieranie wartosci atrybutów) oraz obsluge notyfikacji.

- warstwa posrednia (ang. agent level) – stanowi element posredni pomiedzy zarzadzanymi zasobami, a aplikacjami zarzadzajacymi. Tworza ja serwery MBean (ang. MBean server), w ramach których dzialaja komponenty MBean oraz uslugi JMX. Serwery MBean umozliwiaja rejestracje, wyszukiwanie oraz udostepnianie komponentów MBean. Polaczenie z dzialajacym na MBean serwerze komponentem MBean odbywa sie z wykorzystaniem adapterów protokolów i konektorów. Adaptery protokolów instalowane sa w serwerze MBean, umozliwiaja komunikacje i zarzadzanie z wykorzystaniem standardowych protokolów np. SNMP. Konektory dzialaja zarówno po stronie serwera MBean jak i aplikacji zarzadzajacej, umozliwiaja komunikacje poprzez okreslony protokól komunikacyjny taki jak RMI, HTTP czy HTTPS. Specyfikacja JMX przewiduje tworzenie wlasnych konektorów i adapterów wykorzystujacych dowolny protokól komunikacyjny. Jest to bardzo silny mechanizm, umozliwia bowiem komunikacje praktycznie z kazdym istniejacym systemem monitorowania.
- warstwa zarzadzania (ang. manager level) – umozliwia podejmowanie dzialan na podstawie stanu zarzadzanych zasobów, obejmuje prezentacje informacji oraz sterowanie zarzadzanymi elementami.



Rys. 2 Komponenty architektury JMX.

JMX wspiera trzy podstawowe modele dystrybucji zdarzen: odpytywanie (ang. pull-mode), raportowanie (ang. push-mode) oraz oparte na zdarzeniach notyfikacje (ang. notification). Architektura JMX umozliwia komponentom MBean rozsylanie zdarzen, zarówno do innych komponentów MBean jak i do aplikacji zarzadzajacej, pelniacej role obserwatora. Aplikacja zarzadzajaca powinna w tym celu implementowac wzorzec projektowy Obserwator [GHJ95], oraz zarejestrowac sie jako odbiorca notyfikacji [SUN02c].

Sposród dostepnych implementacji JMX nalezy wymienic opracowana przez SUN Microsystems, rozprowadzana na zasadach komercyjnych JDMK (Java Dynamic Management Kit) oraz dwie bezplatne implementacje XMOJO oraz MX4J. Warto tez nadmienic, ze wersja 1.5 jezyka Java posiada wbudowana implementacje JMX, która jest czescia dystrybucji tego jezyka. Zaimplementowano w niej rozwiazania umozliwiaja monitorowanie i zarzadzanie parametrami maszyny wirtualnej [SUN02c] [MERQ04].

Okreslona w rozwinieciu tezy niniejszej pracy definicja zasobu, jako dowolnego urzadzenia badz aplikacji udostepniajacej interfejs programistyczny, umozliwiajacego odczyt parametrów okreslajacych jego stan, stawia przed autorem bardzo zlozone zagadnienie pozyskania danych. Rozwiazaniem jest wykorzystanie technologii JMX, dzieki której mozna zapewnic uniwersalny interfejs dostepu do dowolnego monitorowanego obiektu. Rozwiazanie takie jest z powodzeniem stosowane do monitorowania heterogenicznych systemów operacyjnych np. przez system JIMS [ZJWB06]. JMX jest równiez wykorzystywany do zarzadzania serwerami aplikacji [LB03], co stanowi wyrazny sygnal do jego wykorzystywania przy konstrukcji systemów komponentowych. Poniewaz niniejsza praca ma przekazac równiez pewne dobre praktyki tworzenia aplikacji, opracowane w jej ramach komponenty beda posiadaly interfejs JMX umozliwiajacy zmiane ich konfiguracji w trakcie dzialania.

2.1.4 Modele informacyjne

Aktualnie wykorzystywane podejscia do konfiguracji i zarzadzania systemami rozproszonymi sa niewystarczajace dla okreslenia waznych celów technicznych i biznesowych oraz utrudniaja adaptacje uslug do róznych wymagan zmieniajacego sie srodowiska dzialania. Przykladowo w SNMP oraz interfejsie konsoli (ang. CLI – ang. command line interface) nie mozna wyrazic regul biznesowych, polityk i procesów w sposób standardowy. Fakt ten praktycznie uniemozliwia bezposrednia zmiane konfiguracji sieci w odpowiedzi na nowe lub zmienione reguly biznesowe. Oprogramowanie musi tlumaczyc

wymagania biznesowe do postaci, która moze byc nastepnie przetlumaczona do SNMP albo CLI. Istnienie szerokiego spektrum dialektów interfejsów konsolowych oraz istotne róznice pomiedzy mozliwosciami poszczególnych wersji sieciowych systemów operacyjnych dodatkowo komplikuje to zagadnienie. Sa to symptomy bardziej powaznego problemu, jakim jest brak definicji wspólnego modelu informacyjnego [Int04].

> **Model informacyjny** (ang. information model) jest abstrakcja i reprezentacja elementów zarzadzanego srodowiska. Zawiera definicje atrybutów, operacji, ograniczen oraz wzajemnych relacji tych elementów. Nie jest zalezny od zadnego specyficznego typu repozytorium, oprogramowania oraz protokolu dostepu [RFC3198] [Str03].
>
> **Model danych** (ang. data model) jest konkretna implementacja modelu informacyjnego (...). Obejmuje struktury danych, operacje i reguly okreslajace w jaki sposób dane sa przechowywane, udostepniane i modyfikowane [RFC3198][Str03].

Problem standaryzacji i integracji mechanizmów monitorowania i zarzadzania systemów rozproszonych jest podejmowany od wielu lat przez szereg instytucji, takich jaki DTMF (ang. Distributed Management Task Force), W3C (ang. Word Wide Web Consortium), ISO/ITU (ang. International Standards Organization / International Telecommunications Union). Organizacje to opracowaly rózne, konkurujace ze soba modele informacyjne i modele danych. Do najwazniejszych nalezy zaliczyc CIM, WBM oraz DEN-ng, zostaly one przyblizone ponizej.

Common Information Model

Jednolity model informacji (ang. CIM – Common Information Model) jest pojeciowym modelem informacyjnym opisujacym elementy sprzetowe, obliczeniowe oraz biznesowe w srodowisku przedsiebiorstwa i Internetu. Schemat CIM (ang. CIM schema) ustala pojeciowo jednolita platforme, opisujaca zarzadzane srodowisko. Podstawowa taksonomia obiektów jest definiowana zarówno z uwzglednieniem klasyfikacji i skojarzen jak i z uwzglednieniem podstawowego zbioru klas tworzacych wspólna platforme [DMTF99] [DMTF00]. CIM jest modelem informacji, pojeciowym obrazem zarzadzanego srodowiska, który ma na celu unifikacje i rozszerzenie istniejacych standardów w zakresie instrumentacji i zarzadzania (np. SNMP) przy wykorzystaniu zorientowanych obiektowo metod projektowania. CIM ma swoje korzenie w podejsciu obiektowym, dzieki któremu model posiada wiele wlasnosci, których brak w innych modelach:

- Abstrakcja i klasyfikacja – W celu zredukowania zlozonosci domeny problemu, zdefiniowane zostaly podstawowe koncepcje, „obiekty” zarzadzanej domeny. Obiekty te sa podzielone na typy - „klasy” poprzez wyróznienie wspólnej charakterystyki i cech (wlasnosci), relacji (asocjacji) i zachowan (metod).
- Dziedziczenie obiektowe – mozliwe jest rozszerzanie istniejacych obiektów. Obiekt rozszerzajacy dziedziczy cala informacje (wlasnosci, metody i asocjacje), okreslona dla obiektów rozszerzanych.
- Zdolnosc do przedstawienia zaleznosci, asocjacji komponentów i polaczen – wykorzystanie przez CIM paradygmatu obiektowego umozliwia bezposrednie modelowanie relacji i asocjacji, które moga byc nazwane i zdefiniowane.
- Dziedziczenie metod standardowych – polega na mozliwosci definiowania standardowego zachowania obiektu (metod). Podejscie takie umozliwia zdefiniowanie metody np. „reset” czy „reboot” i stosowanie jej do dowolnego systemu komputerowego bez wzgledu na jego system operacyjny czy model sprzetowy.

CIM umozliwia modelowanie róznych aspektów zarzadzanego srodowiska, a nie jedynie pojedynczej przestrzeni problemowej. Obejmuje on definicje szeregu jednolitych modeli dla róznych obszarów problemowych. Wywodza sie one z obiektów i koncepcji podstawowych zdefiniowanych w modelu rdzeniowym (ang. Core Model). Z uwagi na swoja zlozonosc CIM nie posiada wielu kompletnych implementacji; do najbardziej znanych naleza systemy adresujace problemy integracji i wymiany danych pomiedzy róznymi systemami zarzadzajacymi – AutevoMDI (IntaMission)[May04] oraz Tivoli (IBM)[IBM04a].

WBEM

WBEM (ang. Web-Based Enterprise Management) jest zbiorem standardów z zakresu zarzadzania i Internetu stworzonych w celu zarzadzania systemami rozproszonymi. WBEM umozliwia tworzenie dobrze zintegrowanych, standaryzowanych narzadzi do zarzadzania, ulatwiajacymi wymiane danych pomiedzy róznymi technologicznie systemami i platformami. Standardy te obejmuja protokoly dostepu (ang. WBEM Protocols), mechanizmy wyszukiwania (ang. WBEM Query), zarzadzania z uzyciem uslug web service (ang. WBEM: WS Management) oraz jezyki zapytan (ang. WBEM Query Languages). WS-Management

jest pierwsza specyfikacja organizacji DTMF umozliwiajaca eksponowanie zgodnych z CIM zasobów jako zbioru uslug web service [DMTF06].

DEN-ng

DEN-ng (ang. Directory Enabled Networks – Next Generation) jest zorientowanym obiektowo modelem informacyjnym, który zapewnia spójny, kompletny i rozszerzalny sposób reprezentacji elementów zarzadzanego srodowiska. Elementy te obejmuje uzytkowników, polityki, urzadzenia, uslugi, protokoly oraz ich konfiguracje. DEN-ng reprezentuje je w sposób abstrakcyjny i definiuje atrybuty, operacje oraz wzajemne relacje w sposób niezalezny od typu repozytorium, oprogramowania oraz protokolu dostepu. Cechami, które wyrózniaja DEN-ng sa:

- wykorzystanie i rozszerzenie UML (ang. Unified Modeling Language) do definiowania modeli danych,
- bazowanie na automacie o skonczonej liczbie stanów, co umozliwia opisywanie cyklu zycia elementów modelu,
- wykorzystanie wzorców oraz rozszerzalnego zbioru ról, które umozliwiaja rozdzielenie bazowej definicji obiektu od tego, w jaki sposób jest on wykorzystywany,
- normalizacja reprezentacji heterogenicznych urzadzen i ich wlasnosci poprzez wprowadzenie abstrakcji bazujacej na okreslaniu: mozliwosci, ograniczen, kontekstów oraz profili urzadzen.

DEN-ng jest zbudowany jako „platforma platform” umozliwiajaca wykorzystywanie i rozszerzanie koncepcji i danych z innych modeli. Ulatwia tworzenie rozwiazan dostosowujacych dzialanie zarzadzanego srodowiska do celów i polityki biznesowej organizacji [Int04].

2.1.5 Wnioski

Istniejace systemy monitorowania dla systemów rozproszonych stanowia bardzo obszerny i róznorodny technologicznie zbiór rozwiazan. Bazuja one na róznych koncepcyjnie i implementacyjnie modelach informacji, modelach danych oraz mechanizmach dystrybucji i dostepu do danych. Mimo ogólnosci przedstawionych modeli informacyjnych, zaden z nich nie zostal bezposrednio wykorzystany do budowy systemu zbierania i przechowywania

danych. Podstawowym powodem jest fakt, ze przedstawione modele zorientowane sa na konkretne obszary zastosowan, co dla rozwazanej klasy systemów nie jest istotne; tworzony system ma bowiem przechowywac dane, a nie je interpretowac. Modele te sa równiez bardzo rozbudowane, co powaznie utrudnia ich wykorzystanie, a celem pracy jest weryfikacja koncepcji budowy systemu w technologiach komponentowych, a nie stworzenie produktu opartego o jeden z przemyslowych standardów.

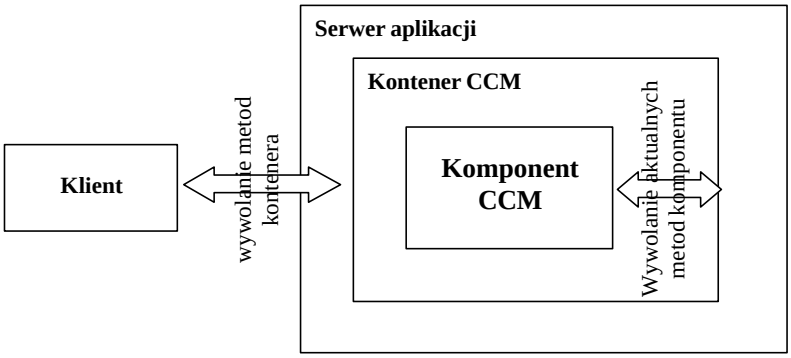
Przedstawiona w punkcie analiza modeli informacji, modeli danych oraz modeli dystrybucji danych jest podstawa dla przedstawienia w rozdziale 3 wymagan, ogólnej architektury, modelu informacyjnego i modelu danych, oraz uniwersalnych interfejsów dostepu do danych dla systemu zbierania i przechowywania danych. Komponentowa budowa systemu jest waznym wymaganiem postawionym w tezie pracy, kwestii wyboru platformy komponentowej poswiecony jest kolejny punkt.

2.2 Technologie komponentowe

Istotnym elementem w procesie projektowania systemu komponentowego jest kwestia wyboru platformy komponentowej, w oparciu o która system bedzie realizowany. Aktualnie dostepne sa trzy rozwiazania: CCM, dotNET oraz J2EE. Przedstawiona w tym punkcie, krótka charakterystyka kazdej z nich jest podstawa dla dokonania wyboru platformy komponentowej do realizacji systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych.

2.2.1 CORBA – CCM

Model komponentowy CORBA (ang. CCM - CORBA Component Model) jest specyfikacja srodowiska do tworzenia skalowalnych, neutralnych jezykowo, transakcyjnych, bezpiecznych, dzialajacych po stronie serwera aplikacji wielkiej skali [Sie01]. Zapewnia on spójna komponentowa rame architektoniczna dla tworzenia rozproszonych aplikacji wielowarstwowych. Stworzone i dzialajace w jej ramach komponenty nazywane sa komponentami CORBA. CCM jest elementem, powstajacej od roku 1999, specyfikacji CORBA 3.0, której wersja 3.0.3 jest aktualnie obowiazujaca [OMG04].



Rys. 3 Architektura CCM.

Typowa architektura CCM wyróznia nastepujac elementy - Rys. 3:

- Serwer aplikacji – zapewnia on srodowiska przetwarzania oraz podstawowe uslugi systemowe, które bazuja na uslugach dostepnych we wczesniejszych specyfikacjach CORBA: przenosnego adaptera obiektów (ang. POA - Portable Object Adapter), brokera zadan obiektów (ang. ORB - Object Request Broker), CORBA Transactions, CORBA Security, CORBA Persistence, CORBA Events, itp.
- Kontener CCM (ang. CCM Container) – dziala jak interfejs pomiedzy komponentami CORBA, a swiatem zewnetrznym; w CCM klient nie posiada bezposredniego dostepu do komponentu.
- Komponenty CCM (ang. CCM Components) – komponenty CCM moga udostepniac wiele interfejsów, których eksponowanie oraz wykorzystywanie jest okreslane na etapie definiowania interfejsu IDL (ang. Interface Definition Language) dla komponentu. Podobnie okreslone musza zostac zdarzenia, które komponent moze przyjmowac lub generowac. CCM wyróznia cztery podstawowe typy komponentów: komponent uslugowy (ang. service component), komponent sesyjny (ang. session component), komponenty procesu (ang. process component) oraz komponent encyjny (ang. entity component).
- Klient CCM – wykorzystuje komponenty CCM.

Sposród najbardziej dojrzalych i stosunkowo kompletnych implementacji CCM mozna wymienic OpenCCM [Mer03] oraz JavaCCM. Niewielka liczba implementacji oraz udostepnianie jedynie czesci funkcjonalnosci wymaganej przez specyfikacje jest spowodowane zaangazowaniem wiekszosci producentów platform komponentowych w walke na rynku serwerów aplikacji J2EE. Specyfikacja CCM nie zostala wybrana przez autora do

implementacji systemu zbierania i przechowywania danych, glównie z powodu malej liczby implementacji, w pelni zgodnych ze specyfikacja.

2.2.2 Rozwiazania firmy Microsoft

Firma Microsoft od wielu lat promuje wlasna grupe standardów zwanych na poczatku OLE/COM, a nastepnie w miare rozwoju, DCOM, COM+, ActiveX oraz DNA [RC98][Lau01], az do wprowadzonego ostatnio terminu i zupelnie nowej koncepcji opartego na komponentach przetwarzania rozproszonego na platformie Microsoft jaka jest dotNET remoting [Str03][OH01].

Microsoft dotNET remoting jest platforma umozliwiajaca współdzialanie obiektów pomiedzy róznymi domenami (zarówno aplikacyjnymi jak i systemowymi czy sprzetowymi). Platforma udostepnia szereg serwisów, sposród których najwazniejsze to: tworzenie obiektów na zadanie (ang. activation), zarzadzanie cyklem zycia oraz transport wiadomosci pomiedzy zdalnymi aplikacjami poprzez kanaly komunikacyjne. DotNET remoting zostal zaprojektowany z uwzglednieniem mechanizmów bezpieczenstwa, które zapewniaja kontrole dostepu do wiadomosci lub zakodowanego obiektu [Ric02]. Ciekawa koncepcja jest zaproponowanie kilku mechanizmów aktywacyjnych, które mozna podzielic na dwie grupy:

- obiekty aktywowane przez klienta (ang. Client-activated objects),
- obiekty aktywowane przez serwer (ang. Server-activated objects).

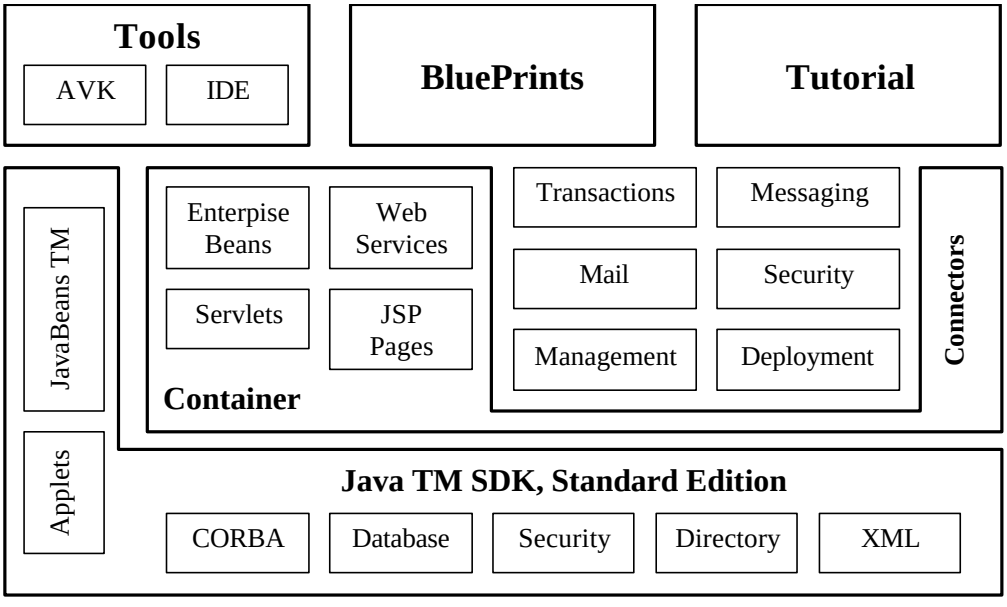
Obiekty aktywowane przez klientów sa pod kontrola, bazujacego na filozofii leasingu, zarzadcy, który wymusza ich usuwanie po wygasnieciu leasingu. Obiekty aktywowane przez serwer moga byc aktywowane w dwóch trybach: pojedyncze wywolanie (ang. single call) lub pojedyncza instancja (ang. singleton) [GHJ95]. Obiekt aktywowany w trybie pojedyncze wywolanie jest usuwany po wywolaniu jednej metody, a w trybie pojedynczej instancji jego czas zycia jest kontrolowany przez mechanizmy bazujace na leasingu.

Stworzenie perfekcyjnej platformy dla rozproszonych obiektów, spelniajacej wymagania wszystkich aplikacji i ich twórców jest z pewnoscia trudne, jesli nie niemozliwe. Stworzenie platformy dotNET remoting, z jej szerokimi mozliwosciami rozszerzania i konfiguracji, jest z pewnoscia krokiem we wlasciwym kierunku. Podkreslenia wymaga fakt wyjscia naprzeciw standardom jakim jest wyposazenie platformy w mechanizmy SOAP/XML, dzieki którym mozliwa jest integracja z innymi, czesto konkurencyjnymi rozwiazaniami [OH01].

Zarówno rozwiazanie dotNET jak i wspomniane wczesniej COM, DCOM bazuja i dzialaja wylacznie w systemie operacyjnym MS Windows. Jest to duze ograniczenie, gdyz wiekszosc systemów monitorujacych pracuje w srodowisku heterogenicznym. Tworzony w ramach niniejszej pracy system powinien dzialac na róznych platformach sprzetowych i róznych systemach operacyjnych. Rozwiazanie firmy Microsoft nie spelnia tych wymagan, nie zostalo zatem wybrane przez autora do stworzenia komponentowego systemu zbierania i przechowywania danych.

2.2.3 Platforma J2EE

Platforma J2EE (*ang. Java 2 Platform, Enterprise Edition*) jest przemyslowym standardem do tworzenia przenosnych, skalowalnych, bezpiecznych, uruchamianych po stronie serwera aplikacji Java. Sklada sie ona z czterech elementów: specyfikacji J2EE, implementacji referencyjnej, zestawu testów zgodnosci oraz materialów dydaktycznych (ang. blueprints) [SUN03a][AA99][RAJ02][SUN02a]. Platforma laczy w sobie bardzo wiele dostepnych w jezyku Java technologii - Rys. 4. Aktualnie najbardziej popularnym standardem jest specyfikacja J2EE 1.4 powstala w roku 2003. Jest juz równiez dostepna wersja specyfikacji 3.0.



Rys. 4 Elementy skladowe technologii J2EE.

Srodowiskiem dzialania dla aplikacji J2EE jest serwer aplikacji. Serwer aplikacji zapewnia aplikacjom uslugi, z których moga korzystac w czasie swojego dzialania, sa to m.in. autoryzacja i autentykacja, zarzadzanie, usluga nazewnicza, dostep do bazy danych. W ramach serwera aplikacji dzialaja kontenery; dwa podstawowe to kontener serwletów i

kontener EJB (ang. Enterprise Java Bean). W kontenerze serwletów przetwarzane sa komponenty WEB, czyli stworzone w technologii JSP strony WWW oraz klasy Java zgodne z Servlet API. Zadaniem kontenera EJB jest zapewnienie srodowiska dzialania dla komponentów EJB, z których zbudowana jest komponentowa aplikacja J2EE. Komponenty EJB sa to klasy i interfejsy napisane w jezyku Java, ukierunkowane na realizacje logiki dzialania aplikacji. Za zarzadzanie cyklem zycia komponentów oraz zapewnieniem dostepu do uslug, z których komponenty korzystaja w czasie dzialania, odpowiada kontener EJB.

Koncepcja zastosowania kontenera komponentów jest bardzo zblizona do wykorzystywanej w CCM – punkt 2.2.1. W istocie specyfikacja komponentów EJB zawiera bowiem elementy specyfikacji CCM. Byc moze wlasnie dzieki przyjetym uproszczeniom istnieje bardzo duza liczba implementacji serwerów aplikacji zgodnych ze specyfikacja J2EE. Jako pierwsza nalezy wymienic, dostarczana przez Sun Microsystems, implementacje referencyjna [ABB04]. Liderami w zakresie popularnosci sa: komercyjna, bardzo wydajna, ale tez i droga, implementacja firmy BEA o nazwie WebLogic oraz dostarczana na licencji open-source implementacja o nazwie JBoss. Dalsze pozycje obejmuja produkty wiekszosci liczacych sie dostawców oprogramowania: Borland – Entrprise Server, Sun - JES (dawniej SUN One i iPlanet), IBM – Websphere, Macromedia – JRun, Novell – exteNd, Oracle – Oracle AS, Sybase – EAServer, Caucho – Resin, ObjectWeb – JonAS by wymienic tylko wazniejsze. Kwestie ich porównania w zakresie mozliwosci, wydajnosci i zgodnosci ze standardem sa szeroko przedstawiane przez wielu autorów [SS05][PMJ01] [MERQ04][Mar01] i nie beda w pracy podejmowane.

2.2.4 Wnioski

Duza uniwersalnosc, ugruntowana pozycja przemyslowego standardu, duza liczba implementacji oraz doswiadczenie autora w tworzeniu aplikacji w technologii J2EE sa powodami, dla których zostala ona wybrana jako technologia budowy komponentowego systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych.

2.3 Wybrane elementy technologii J2EE

W tym punkcie przedstawiono wybrane elementy technologii J2EE. Przyblizona zostala koncepcja warstwowej budowy aplikacji J2EE; wyróżniono i opisano piec podstawowych warstw. Przedstawiona zostala równiez, stosowana w J2EE, koncepcja ukrytego oprogramowania posredniczacego (ang. implicit middleware) wraz ze sposobem jej

wykorzystania w dostepie do komponentów. W kolejnych punktach opisane zostaly dostepne w technologii J2EE typy komponentów; ich wlasnosci funkcjonalne oraz niefunkcjonalne zostaly zebrane w tabelach.

2.3.1 Warstwowy model aplikacji J2EE

Podzial systemu na warstwy jest jednym z najbardziej znanych wzorców architektonicznych [Lar02][AA99]. Opisuje on podzial zlozonego systemu na warstwy logiczne (ang. layers) i jest równiez nazywany podzialem poziomym systemu (ang. horizontal approach). Liczba i rodzaj warstw, na które dzieli sie system jest zalezna od domeny zastosowan systemu. Aplikacje budowane w oparciu o technologie J2EE skladaja sie najczesciej z pieciu warstw logicznych [ACM01][Mar01][KB02], sa nimi:

- warstwa prezentacji (ang. presentation layer) – jest warstwa graficznego interfejsu uzytkownika, zbudowana w praktycznie dowolnej technologii, np. HTML, JSP, WML czy Flash.
- warstwa aplikacji (ang. application layer) – to warstwa przeplywu zadan i sterowania miedzy elementami warstwy interfejsu graficznego uzytkownika, a warstwa biznesowa aplikacji. Odpowiada ona za utrzymywanie stanu sesji klienta, dokonywanie walidacji syntaktycznej wprowadzanych danych oraz delegowanie zadan do warstwy biznesowej w celu wykonania logiki aplikacji.
- warstwa biznesowa (ang. business layer) – jest nazywana równiez warstwa uslug (ang. services layer) oraz warstwa procesów i przeplywu sterowania (ang. process and workflow layer). Stanowi miejsce dostepu do logiki biznesowej systemu. Elementy warstwy aplikacji, chcac zrealizowac okreslony przypadek uzycia, wywoluja odpowiednie metody biznesowe na komponentach warstwy biznesowej. W przypadku aplikacji tworzonej w technologii J2EE warstwa biznesowa jest zaimplementowana zgodnie ze wzorcem projektowym fasady sesyjnej [RAJ02]. Do glównych zadan tej warstwy nalezy wykonywanie logiki biznesowej aplikacji poprzez odwolywanie sie do obiektów warstwy domeny systemu, kontrolowanie przebiegu transakcji oraz zarzadzanie bezpieczenstwem wykonywanych operacji biznesowych.
- warstwa domeny (ang. domain layer) – jest miejscem dzialania obiektów domeny systemu, zidentyfikowanych na etapie analizy obiektowej. W przypadku aplikacji

opartej o technologie J2EE warstwe te stanowia komponenty encji. Warstwa biznesowa realizuje swoja funkcje dostawcy logiki biznesowej i danych dla strony klienta poprzez odwolywanie sie do obiektów warstwy domeny. Warstwa domeny jest najczesciej na tyle ogólna, ze stanowi zbiór obiektów niezaleznych od wykorzystujacej ja aplikacji i w zwiazku z tym moze zostac ponownie uzyta przy tworzeniu innych systemów.

- warstwa integracji (ang. integration layer) – nazywana jest równiez warstwa zródla danych (ang. data source layer), warstwa dostepu do danych (ang. data access layer), warstwa persystencji (ang. persistence layer) lub po prostu warstwa danych (ang. data layer). Odpowiada za pobieranie i umieszczanie obiektów domeny w zródle danych (ang. data source), którym najczesciej jest relacyjna bazie danych.

Dodatkowo wyróznia sie warstwe podstawowa:

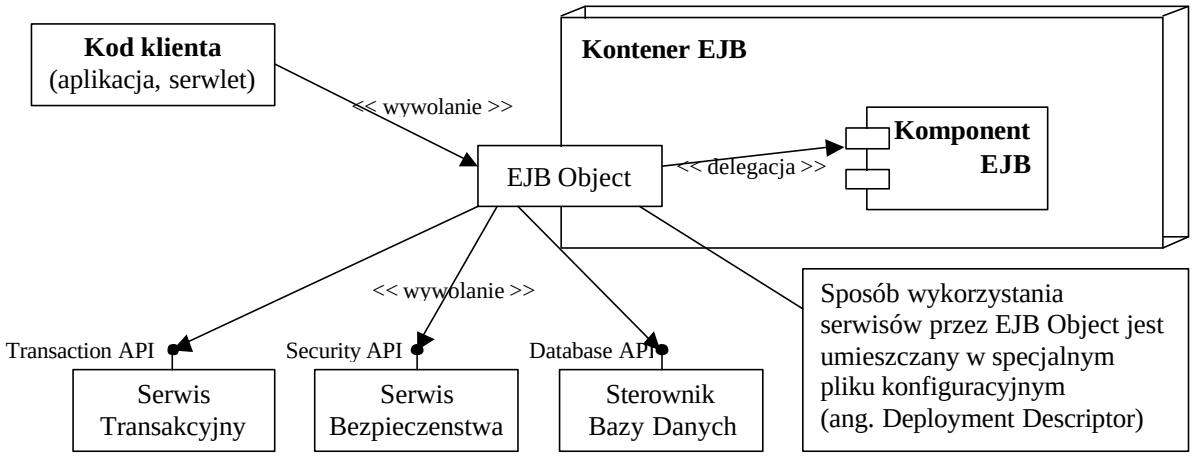
- warstwa podstawowa (ang. foundation layer) – nazywana jest równiez warstwa techniczna (ang. technical layer), warstwa infrastruktury (ang. infrastructure layer) lub warstwa wspólna (ang. common layer). W jej sklad wchodzi kod obslugi bledów systemu, jego konfiguracji, logowania, a takze klasy pomocnicze (ang. utilities) systemu. Wszystkie te serwisy uzywane sa w calej aplikacji przekraczajac granice warstw logicznych. Dlatego tez bardzo czesto nazywane sa kodem niezaleznym od warstw.

Warstwy aplikacji i prezentacji nosza wspólna nazwe warstwy interfejsu uzytkownika (ang. UI layer) lub warstwy klienta (ang. client layer). Uzyciem modelu warstwowego kieruja dwie przeciwstawne zasady. Pierwsza z nich mówi, iz kazdy problem moze zostac rozwiazany przez wprowadzenie dodatkowej warstwy posredniej, druga stwierdza, ze kazda nowa warstwa wprowadzona do systemu obniza szybkosc i sprawnosc jego pracy. Prawdziwa trudnosc lezy we wlasciwym wywazeniu obu czynników i zaprojektowaniu systemu o akceptowalnej zlozonosci i wydajnosci [RR02].

2.3.2 Komponenty Enterprise Java Beans

Elementem technologii J2EE jest specyfikacja EJB. Najwazniejszym elementem specyfikacji EJB sa komponenty EJB (ang. Enterprise Java Beans). Sa to programowe komponenty realizujace logike dzialania aplikacji, dzialajace oraz udostepniane w ramach kontenera komponentów. Obiekty (klienci) wykonujace operacje na komponencie EJB, nigdy nie lacza sie z komponentem bezposrednio. Kontener wykorzystuje mechanizm intercepcji,

realizowany w oparciu o obiekt posredniczacy *EJB Object*, który przechwytuje wywolania operacji. Po przechwyceniu, wykonywane sa, okreslone w pliku konfiguracyjnym komponentu (ang. deployment descriptor), odwolania do serwisów, obejmujace np. rozpoczecie transakcji albo weryfikacje praw dostepu. Nastepnie operacje delegowane sa do wlasciwej metody komponentu - Rys. 5. Architektura EJB zostala zaprojektowana w taki sposób, aby odwolania do serwisów oraz istnienie posrednika pomiedzy klientem, a komponentem bylo dla obu stron niewidoczne. Jest to zgodne z koncepcja ukrytego oprogramowania posredniczacego (ang. implicit middleware).



Rys. 5 Mechanizm intercepcji w dostepie do komponentu EJB.

Komponenty skladaja sie z klasy zawierajacej atrybuty i metody komponentu oraz zestawu interfejsów. Interfejsy komponentu moga byc dwojakiego rodzaju: lokalne (ang. local) oraz zdalne (ang. remote). Lokalne interfejsy komponentu sa wykorzystywane do lokalnej (w ramach jednej maszyny wirtualnej) komunikacji z komponentem, zdalne sa wykorzystywane do komunikacji zdalnej. Dodatkowo komponent musi posiadac interfejs domowy (ang. home), który jest wykorzystywany do tworzenia, wyszukiwania oraz usuwania komponentów. Zarówno klasa komponentu jak i interfejsy musza, zgodnie ze specyfikacja, rozszerzac odpowiednie interfejsy bazowe, co umozliwia kontenerowi jednolite zarzadzanie cyklem zycia komponentu [Hae01].

Proces tworzenia komponentów i deskryptorów oraz skladania z nich aplikacji jest zlozony. Dostepnych jest szereg graficznych srodowisk programistycznych (ang. IDE - Integrated Development Environment) upraszczajacych ten proces np. : JES - Java Enterprise Studio, PowerDesigner, NetBeans i WebLogicPlatform. Narzedzia te sa tworzone przez producentów serwerów aplikacji i wspieraja glównie mozliwosci ich wlasnych serwerów. Alternatywna dla tych narzadzi jest rozwiazanie o nazwie XDoclet [WAC+03][WR03]. Bazuje ono na umieszczaniu w kodzie zródlowym komponentu dodatkowych informacji, które sa

wykorzystywane do automatycznego generowania interfejsów komponentów oraz deskryptorów. Sposób ten zostal wykorzystany do implementacji fragmentów systemu. Specyfikacja EJB wyróznia trzy typy komponentów: komponent sesyjny, komponent encyjny oraz komponent sterowany zdarzeniami; zostaly one przedstawione w kolejnych punktach.

Komponenty sesyjne

Komponenty sesyjne (ang. session bean) sa specjalizowanymi obiektami biznesowymi przeznaczonymi do indywidualnej obslugi klientów; reprezentuja obiekty i procesy o krótkim czasie zycia (ang. transient). Specyfikacja wyróznia dwa typy komponentów sesyjnych: stanowe (ang. statefull) i bezstanowe (ang. stateless). Stanowe komponenty sesyjne zachowuja swój stan pomiedzy kolejnymi, wywolywanymi przez klienta operacjami. Bezstanowe komponenty sesyjne sa zorientowane na przetwarzanie pojedynczych operacji i nie zachowuja stanu pomiedzy kolejnymi wywolaniami operacji. Decyzja o wyborze typu komponentu wynika glównie z analizy funkcjonalnej tworzonego systemu. Poza czynnikami funkcjonalnymi nalezy jednak uwzglednic wplyw wyboru na wydajnosc i skalowalnosc. Komponenty bezstanowe sa uznawane za najlzejsze i najlepiej skalowalne, natomiast komponenty stanowe, za duzo ciezsze i trudniej skalowalne.

Wlasnosc	**Stanowy komponent sesyjny**		**Bezstanowy komponent sesyjny**
	Aktywny mechanizm replikacji sesji	**Brak mechanizmu replikacji sesji**	
Zachowanie stanu miedzy wywolaniami	✓	✓	–
Przelaczanie wywolan klienta w celu równowazenia obciazenia	Do dowolnego serwera	Wywolania w ramach jednej sesji do tej samej instancji serwera	Do dowolnego serwera
Przezroczyste przejmowanie wywolan w przypadku awarii wezla	✓	–	✓
Transakcyjnosc operacji	Wymaga rozproszonych mechanizmów synchronizacji transakcji	Lokalne zarzadzanie transakcjami	Lokalne zarzadzanie transakcjami
Wplyw na wydajnosc	Zachowywanie i replikacja stanu sesji obniza wydajnosc	Zachowywanie stanu sesji obniza wydajnosc	Brak dodatkowych narzutów
Wplyw na skalowalnosc	Replikacja stanu sesji na wszystkich wezla utrudnia skalowalnosc	✓	✓
Latwa konfiguracja	Replikacja sesji na wybranych wezlach wymaga wprowadzenia mechanizmów sub-partycji, które sa rzadko wspierane	✓	✓

Tabela 1 Wlasnosci kompo nentów sesyjnych.

Zestawienie istotnych wlasnosci obu typów komponentów zawiera Tabela 1. Wyróznia ona wlasnosci stanowych komponentów sesyjnych w zaleznosci od stosowania mechanizmu replikacji sesji. Rozróznienie to jest wazne, bowiem mechanizm replikacji stosowany jest czesto w przypadku dzialania aplikacji na klastrze serwerów aplikacji. Definicja klastra, mozliwosci skalowania i klasteryzacji aplikacji J2EE zostanie szerzej omówione w punkcie 2.4.2.

Komponenty encyjne

Komponenty encyjne (ang. entity bean) sa uzywane do reprezentowania trwalych danych (ang. persistent data), przechowywanych w zródle danych, jakim najczesciej jest relacyjna baza danych. Za synchronizowanie stanu konkretnej instancji komponentu z baza danych odpowiada kontener. Wywoluje on odpowiednie metody synchronizujace ilekroc na komponencie wykonywana jest operacja dostepu (zapisu/odczytu) do danych. Metody synchronizujace moga byc zaimplementowane w komponencie lub zapewniane przez kontener. Przypadek, kiedy komponent implementuje metody synchronizujace nazywany zostal BMP (ang. Bean-Managed Persistence). Kiedy za implementacje metod odpowiada kontener, mamy do czynienia z komponentem typu CMP (ang. Container-Managed Persistence). W podejsciu BMP programista musi implementowac kod zapewniajacy synchronizacje stanu komponentu z baza danych. Tworzenie takiego kodu jest bardzo pracochlonne i moze pociagac za soba problemy z przenosnoscia kodu komponentu miedzy róznymi bazami danych, bywa jednak koniecznoscia w przypadku, gdy zródlem danych dla komponentu jest system zastany lub baza danych nie wspierajaca standardu JDBC. W podejsciu CMP implementacja metod synchronizujacych nie jest potrzebna – sa one generowane automatycznie na podstawie deskryptora komponentu, w którym okresla sie sposób odwzorowania atrybutów komponentu na kolumny w tabeli bazy danych. Róznice obu podejsc zbiera Tabela 2.

Zlozony problem odwzorowania obiektów na rekordy w relacyjnej bazie danych jest opisywany przy pomocy deskryptora komponentów. W deskryptorze tym znajduja sie równiez definicje metod sluzacych do wyszukiwania komponentów encyjnych. Definicje sa budowane z wykorzystaniem jezyka EJB-QL (ang. EJB Query Language), który jest ubogim, obiektowym odpowiednikiem jezyka SQL.

Komponent / Wlasnosci	CMP	BMP
Zarzadzanie zapisem	Kontener	Komponent
Transakcyjnosc operacji	Okreslana w deskryptorze	Okreslana w deskryptorze lub kodowana w komponencie
Buforowanie odczytu	✓	Zalezy od implementacji
Buforowanie zapisu	–	–
Automatyczne generowanie komend SQL	✓	–
Metody dostepu do pól	Generowane automatycznie	Kodowane w komponencie
Metody do wyszukiwania komponentów	Obslugiwane przez kontener, kodowane w EJB QL	Obslugiwane przez komponent, kodowane w SQL
Wsparcie dla relacji pomiedzy komponentami	Deklaratywne	Kodowane w komponencie
Tworzenie wlasnych wywolan SQL	–	✓
Operacje grupowe SQL	–	–
Wsparcie dla róznych baz danych	✓	Zalezy od implementacji

Tabela 2 Wlasnosci komponentów warstwy domeny.

Specyfikacja okreslajaca skladnie deskryptora komponentu umozliwia podanie jedynie bardzo podstawowych parametrów dzialania komponentu, sposobu odwzorowania i wyszukiwania, dlatego wszyscy producenci serwerów aplikacji wprowadzili dodatkowo specyficzne dla swoich rozwiazan deskryptory. Definiuje sie w nich specyficzne dla konkretnej implementacji serwera aplikacji informacje, które w istotny sposób wplywaja na funkcjonalnosc oraz wydajnosc dzialania komponentu. Fakt ten powoduje, ze przenosnosc komponentów pomiedzy róznymi serwerami jest znaczaco utrudniona.

Komponenty sterowane komunikatami

Komponenty sterowane komunikatami (ang. Message Driven Bean) sluza do asynchronicznego przetwarzania komunikatów. Przetwarzanie asynchroniczne w architekturze J2EE bazuje na specyfikacji JMS (ang. Java Messaging Service). Specyfikacja ta okresla interfejsy do tworzenia, nadawania i odbierania wiadomosci, definiuje modele komunikacji oraz gwarantuje pewnosc dostarczenia informacji [SUN03c]. Komponenty sterowane komunikatami sa odbiorcami komunikatów. Specyfikacja EJB wymaga aby komponent sterowany komunikatami implementowal interfejs *MessageListener*. Komponenty

sa tworzone i usuwane przez kontener, który dostosowuje ich liczbe do aktualnych potrzeb oraz dostepnych zasobów. Po stworzeniu, komponent jest rejestrowany jako odbiorca komunikatu dla konkretnej kolejki JMS, i oczekuje na przychodzace w sposób asynchroniczny komunikaty. Nadawca komunikatu moze byc dowolny obiekt aplikacji dzialajacej w ramach serwera J2EE lub dowolna aplikacja zewnetrzna. Nadawca tworzy komunikat i przesyla go do brokera komunikatów, czyli dostarczonej wraz z serwerem aplikacji implementacji JMS. Nastepnie broker przekazuje wiadomosc do kontenera EJB, który wywoluje metode *onMessage()* komponentu sterowanego komunikatami.

W przeciwienstwie do komponentów encyjnych i sesyjnych, komponenty sterowane komunikatami nie wymagaja dodatkowych, specyficznych dla producentów, deskryptorów konfiguracji, gdyz parametry uwzglednione przez specyfikacje EJB sa wystarczajace. Specyficzna konfiguracja dotyczy jednak implementacji i konfiguracji brokerów komunikatów implementujacych API JMS. Standard okresla bowiem jedynie interfejsy komunikacyjne, pozostawiajac pelna dowolnosc ich implementacji oraz udostepnianych mechanizmów konfiguracji.

2.3.3 Wnioski

Architektura J2EE daje duze mozliwosci w zakresie tworzenia wydajnych, skalowalnych aplikacji komponentowych o zlozonej funkcjonalnosci. Wspiera koncepcje warstwowej budowy aplikacji oraz udostepnia kilka typów komponentów dla róznych warstw.

System zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych zostanie zbudowany w warstwie biznesowej w oparciu o sesyjne komponenty bezstanowe, a w warstwie domeny o komponenty encyjne typu CMP. Podejscie takie upraszcza uzyskanie, postulowanej w tezie pracy wlasnosci skalowalnosci systemu, oraz zapewni przenosnosc systemu pomiedzy róznymi bazami danych. W dalszej czesci tego rozdzialu przedstawiona zostala koncepcja wykorzystania komponentów sterowanych komunikatami w celu poprawy wydajnosci dzialania systemu.

Opisane w tym punkcie wybrane elementy technologii J2EE, stanowia wprowadzenie w problematyke budowy aplikacji w tej technologii. Autor przedstawil jedynie bardzo podstawowe i wazne dla wykazania prawdziwosci tezy pracy aspekty technologii J2EE, swiadomie pomijajac bardzo wiele mechanizmów, których znajomosc jest potrzebna do efektywnego projektowania i implementacji aplikacji. Szersze informacje Czytelnik moze

znalezc w bardzo bogatej literaturze z tego zakresu np.: [Hae01][CC04][SUN04c][Glo04] [OF02][Cen04] oraz kolejnym punkcie, poswieconym technikom zwiekszania wydajnosci systemów komponentowych.

2.4 Techniki zwiekszania wydajnosci systemów komponentowych

Dla potrzeb niniejszej pracy wydajnosc systemu komputerowego zostala okreslana jako maksymalna mozliwa przepustowosc systemu, przy której zachowuje on zadane parametry jakosciowe (ang. QoS - Quality of Service). Dla systemów komponentowych, które najczesciej sa zorientowane na przetwarzanie wywolan operacji, parametry jakosciowe obejmuja czas odpowiedzi systemu (ang. response time) oraz poprawnosc wykonania operacji, natomiast przepustowosc jest mierzona iloscia przetworzonych wywolan w jednostce czasu. Zwiekszanie wydajnosci systemów komponentowych polega na zwiekszaniu ilosci przetwarzanych przez system w jednostce czasu wywolan, przy utrzymaniu czasu odpowiedzi systemu ponizej ustalonej wartosci oraz zachowaniu poprawnosci wykonania operacji.

Wydajnosc systemów komponentowych moze byc zwiekszana poprzez zastosowanie róznych technik. W tym punkcie autor przedstawi rozwiazania najbardziej interesujace z punktu widzenia realizacji tezy pracy. Przyblizone zostaly techniki bazujace na optymalizacji i strojeniu serwera aplikacji oraz komponentów, klasteryzacji serwera aplikacji, optymalizacji zapisu danych w bazie danych oraz wykorzystaniu brokera komunikatów.

2.4.1 Strojenie serwera aplikacji i optymalizacja komponentów

Strojenie serwera aplikacji, a wiec zmiana parametrów konfiguracyjnych systemu w celu uzyskania lepszej wydajnosci, jest procesem bardzo zlozonym. Strojeniu podlegaja nie tylko parametry serwera aplikacji, ale równiez systemu operacyjnego, wirtualnej maszyny Java oraz systemów wspólpracujacych: bazy danych, systemów zastanych i brokerów komunikatów. Odpowiednio przeprowadzone strojenie moze o rzedy wielkosci poprawic wydajnosc systemu bez potrzeby rozbudowy infrastruktury sprzetowej.

Praktycznie z kazdym serwerem aplikacji, dostarczane sa szczególowe wytyczne odnosnie konfiguracji systemu operacyjnego oraz wirtualnej maszyny Java. Wytyczne te uwzgledniaja rózne systemy operacyjne, na których dzialac moga serwery aplikacyjne, oraz precyzuja ustawienia charakterystyczne dla tych systemów, takie jak parametry pracy jadra czy specyficzne ustawienia protokolów TPC/IP. Producenci serwerów aplikacji zalecaja

konkretne wersje maszyny wirtualnej, w oparciu, o która dzialac ma serwer aplikacji, oraz specyficzne ustawienia tej maszyny obejmujace np.: wielkosc i sposób zarzadzania pamiecia oraz konfiguracje procesu odzyskiwania zasobów (ang. garbage collection) [IBM04b] [SUN03d] [SL03].

Dokumentacja obejmuje równiez opis duzej ilosci parametrów serwera aplikacji i kontenera EJB, wplywajacych na wydajnosc uruchamianych aplikacji. Zarówno sposoby dobierania optymalnych wartosci, jak i nazwy konkretnych parametrów sa specyficzne dla róznych implementacji serwerów aplikacji. Dodatkowo, wysoka zlozonosc serwerów aplikacji i bardzo duze zaleznosci pomiedzy parametrami powoduja, ze mimo szczegółowych opisów, uzyskiwanie optymalnych wartosci wymaga licznych eksperymentów i w duzej mierze bazuje na doswiadczeniu administratora.

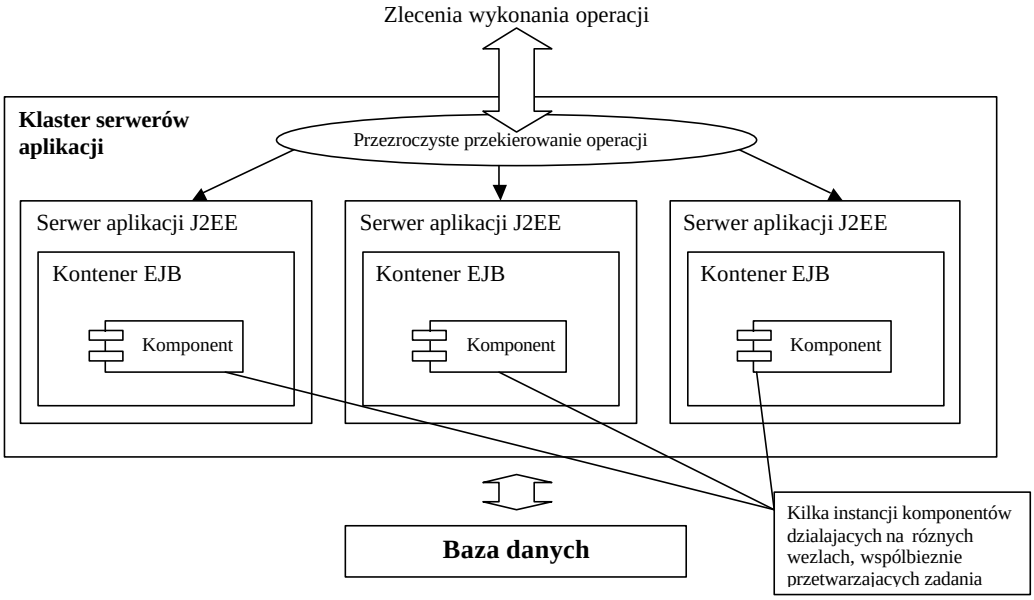
Uzyskanie pozadanej wydajnosci warunkowane jest równiez przez poprawne zaprojektowanie i implementacje komponentów. Tworzone komponenty nalezy budowac w oparciu o uznane wzorce projektowe, dobre praktyki projektowania i kodowania aplikacji w jezyku Java oraz wlasciwa konfiguracje komponentu w pliku deskryptora. Zagadnienia te sa szeroko przedstawione w licznej literaturze poswieconej tworzeniu wydajnych rozwiazan w architekturze J2EE [RSB05] [AA99] [ACM01][Mar02] [KB02].

2.4.2 Klasteryzacja aplikacji

Dla aplikacji dzialajacych w oparciu o jeden serwer zwiekszenie wydajnosci realizowane jest w pierwszym rzedzie poprzez powiekszanie pamieci operacyjnej komputera, instalowanie wydajniejszego lub – jesli to mozliwe – kolejnego procesora; mówimy wtedy o skalowalnosc wertykalnej. Rozwiazania takie prowadza jednak stosunkowo szybko do osiagniecia granicy obciazenia pojedynczej maszyny. Dotyczy to zarówno samego przetwarzania jak i np. przepustowosci interfejsów sieciowych. Dalszy wzrost wydajnosci systemu wymaga zrównoleglenia przetwarzania w oparciu o kilka maszyn tworzacych klaster.

> **Klaster** - jest zbiorem komputerów - wezlów (ang. node), które realizuja wspólny cel, widzianych z zewnatrz jako jeden spójny system (ang. SSI - single system image).

Podstawowym sposobem zwiekszania wydajnosci systemów komponentowych, poza optymalizacja dzialania serwera aplikacji oraz komponentów, jest ich klastryzacja - Rys. 6.



Rys. 6 Klastryzacja komponentów w technologii J2EE.

Klastryzacja systemu komponentowego polega na uruchomieniu kilku instancji serwerów aplikacji (wezlów) oraz takiej konfiguracji tych serwerów, która umozliwi ich współdzialanie przy wykonywaniu zlecanych operacji [Kan01]. Zrównoleglenie przetwarzania jest uzyskiwane poprzez instalowanie i uruchamianie wielu wspólbieznie dzialajacych instancji komponentów na serwerach aplikacji tworzacych klaster - Rys. 6.

W ramach klastra komponentów uruchamianych jest wiele instancji serwerów aplikacji, na których dziala ten sam, badz rózny zbiór komponentów. W technologii J2EE wiekszosc zagadnien konfiguracyjnych zwiazanych z rozmieszczeniem komponentów na klastrze jest okreslana w deskryptorach konfiguracji, dzieki czemu nie jest konieczna modyfikacja kodu komponentów zwiazana z ich uruchomieniem na klastrze [Yu05].

Poza zwiekszeniem wydajnosci systemu, klastryzacja moze poprawic równiez inne wlasnosci systemu, sa to:

- **Wysoka dostepnosc** (ang. high availability) – czyli zdolnosc do przyjmowania wywolan na dowolnym wezle bez wzgledu na stan pozostalych wezlów, dzieki czemu aplikacja pozostaje dostepna pomimo niedostepnosci niektórych wezlów,
- **Równowazenie obciazenia** (ang. load balancing) - zapewnienie efektywnego przetwarzanie wywolan poprzez ich rozdzielenie pomiedzy dostepne w klastrze wezly,

- **Reakcja na bledy** (ang. fail-over) – wlasciwa reakcja na uszkodzenie wezla, polegajaca na automatycznym, niedostrzegalnym dla klienta przekierowaniu wywolan do innych sprawnych wezlów,
- **Heterogenicznosc wezlów** (ang. nodes heterogeneity) – mozliwosc laczenia w klastrze wezlów dzialajacych w oparciu o rózne platformy sprzetowe i systemowe,
- **Dynamiczna rekonfiguracja** (ang. dynamic reconfiguration) – mozliwosc dynamicznej modyfikacji struktury klastra poprzez dodawania lub usuwanie wezlów bez potrzeby chociaz chwilowego przerywania pracy calego systemu,
- **Zintegrowane zarzadzanie** (ang. manageability) – istnienie w systemie pojedynczej konsoli zarzadzajacej, umozliwiajacej kontrole wszystkich elementów systemu oraz dajacej ogólny obraz stanu systemu.

Aby współdzialanie serwerów aplikacji tworzacych klaster bylo mozliwe, musza one posiadac mechanizmy wykrywania innych serwerów aplikacji, mechanizmy zapewniajace wlasciwe przelaczanie operacji pomiedzy serwerami aplikacji oraz globalna usluge nazewnicza. Mechanizmy te zostaly omówione w kolejnych punktach.

Dynamiczna rekonfiguracja klastra

Rekonfiguracja klastra polega na zmianie jego struktury poprzez dodawanie lub usuwanie serwera aplikacji wchodzacego w jego sklad. Zdolnosc systemu do dynamicznej rekonfiguracji oznacza, ze operacja dodania lub usuniecia moze byc dokonywana bez potrzeby chociaz chwilowego przerywania pracy calego systemu. Wymaga to zdolnosci serwerów aplikacji do wykrywania nowych wezlów oraz wlasciwej reakcji na sytuacje ‘zniknieciа’ wezla. Podstawa dla tego procesu jest komunikacja grupowa [KB96].

Komunikacja grupowa w ramach klastra jest realizowana z wykorzystaniem zamknietych standardów dostawców serwerów aplikacji albo poprzez dostepny na otwartej licencji (ang. open source) mechanizm JGroups [Ban98][Li03]. JGroups jest rozbudowana biblioteka zapewniajaca niezawodna komunikacje w grupie, dzieki której stosunkowo latwo mozna reagowac na pojawianie sie i opuszczanie grupy przez wezel. Jest ona wykorzystywana również w celu propagowania informacji o stanie wezlów, wykonywaniu operacji na kilku wezlach oraz wykrywania zakleszczen (ang. deadlock detection). Dynamiczna rekonfiguracja, mimo iz jest wspierana przez wiodace serwery aplikacji, nie jest objeta zadna standaryzacja.

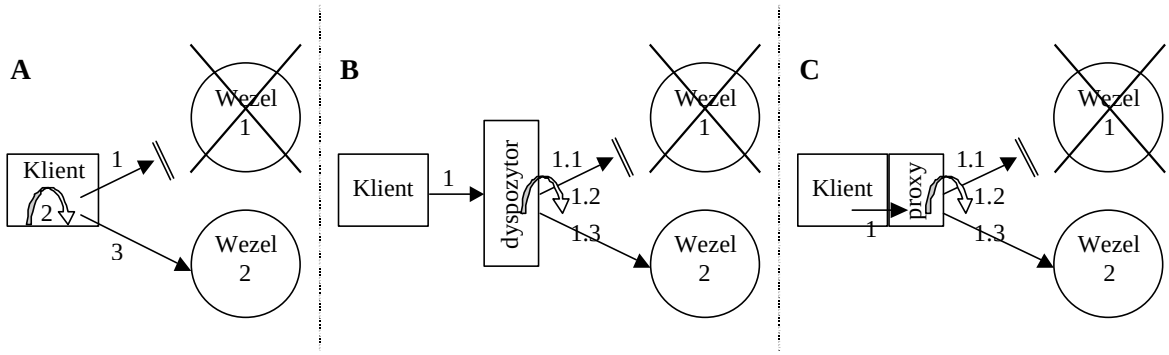
Dlatego nie jest aktualnie mozliwe współdzialanie w obrebie jednego klastra serwerów aplikacji róznych producentów.

Niektóre implementacje serwerów aplikacji posiadaja dodatkowo mechanizm partycji [LB03][SKM+05]. Partycja jest rozumiana jako grupa wezlów tworzacych w klastrze rodzaj podzbioru, w ramach którego dzialaja te same komponenty. Podejscie takie ulatwia dystrybucje komponentów na wybrane wezly, zarzadzanie nimi oraz umozliwia elastyczne przydzielanie zasobów do realizacji okreslonej funkcjonalnosci. Mechanizmy te równiez nie sa objete standaryzacja.

Sposobom automatycznego organizowania sie serwerów aplikacji w klastry oraz mozliwym optymalizacjom tego procesu mozna poswiecic oddzielna prace. Autor poprzestanie na wykorzystaniu istniejacych rozwiazan, pomijajac szczegóły zwiazane z ich implementacja.

Mechanizmy przelaczanie operacji

Zrównoleglenie przetwarzania, w ramach serwerów aplikacji tworzacych klaster, wymaga uzycia mechanizmów przelaczajacych przychodzace do systemu wywolania, do róznych serwerów aplikacji. Mechanizmy te powinny zapewniac równowazenie obciazenia oraz reakcje na awarie wezla. Mozliwe rozwiazania w tym zakresie przedstawia Rys. 7.



Rys. 7 Mechanizmy przelaczania operacji pomiedzy wezlami.

Proces przelaczania do innego (sprawnego, mniej obciazonego) wezla moze byc realizowany: (A) bezposrednio przez aplikacje klienta, (B) poprzez dedykowany, współdzielony element przelaczajacy lub (C) jako inteligentny element posredniczacy (ang. smart proxy), bedacy czescia, dzialajacych po stronie klienta, mechanizmów warstwy posredniej, odpowiedzialnych za komunikacje z serwerem. Sposród tych trzech rozwiazan, ostatnie ma najwiecej zalet. Jest przezroczyste dla klienta oraz nie zawiera pojedynczego współdzielonego elementu dyspozytora (ang. dispatcher), który jest potencjalnym waskim gardlem, i którego awaria calkowicie uniemozliwia dostep do systemu.

Zastosowanie rozwiazania opartego na przelaczaniu, realizowane przez element posredniczacy, jest mozliwe dzieki wlasnosciom jezyka Java oraz mechanizmom zdalnej komunikacji opartej na RMI (ang. Remote Method Invocation), polegajacym na pobieraniu referencji do obiektu w postaci zserializowanego obiektu posredniczacego (ang. stub). Obiekt ten moze byc generowany i pobierany dynamicznie, w trakcie dzialania aplikacji (nie musi byc czescia biblioteki klienta), oraz moze obejmowac specyficzne funkcje, które beda wykonywane po stronie klienta, zapewniajac równowazenie obciazenia i reakcje na bledy [SUN04a]. Obiekt posredniczacy o takich wlasnosciach jest nazywany obiektem posredniczacym swiadomym zwielokrotnienia (ang. replica-aware stub).

W aktualnej implementacji, o nazwie HA-RMI (ang. High Availability RMI), pobierany przez klienta obiekt posredniczacy zawiera liste dostepnych wezlów oraz obiekt implementujacy polityke równowazenia obciazenia [LB03]. Umozliwia to wybór polityk w momencie uruchomienia serwera, a nawet jej przelaczenie w trakcie dzialania systemu. W przypadku zmiany topologii klastra (dodanie, usuniecie, awaria wezla) przy kolejnym zdalnym wywolaniu metody przez klienta, wraz z odpowiedzia przekazywana jest aktualna lista dostepnych wezlów, która jest podstawa dla realizowania odpowiedniej polityki w nastepnych wywolaniach. Dostepne polityki równowazenia obciazenia obejmuja dwie implementacje: algorytm cykliczny (ang. round robin) oraz pierwszy dostepny (ang. first available); dobrze zdefiniowane API umozliwia tworzenie wlasnych implementacji polityk.

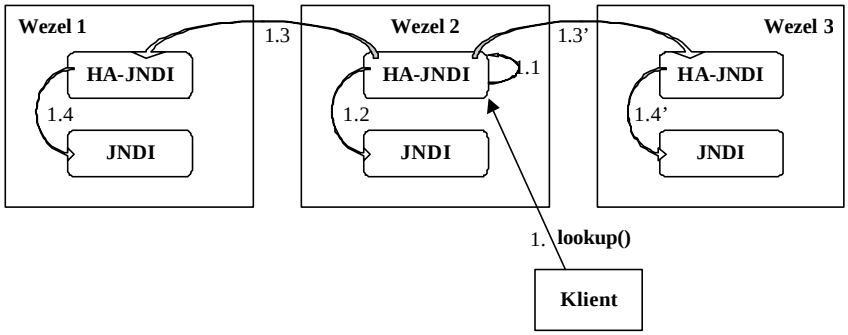
Wysoko dostepna usluga nazewnicza

Klastrowanie serwerów aplikacji pozostanie bezuzyteczne, w przypadku braku wspólnej dla wszystkich wezlów uslugi nazewniczej. Usluga taka, o nazwie HA-JNDI [LB03], jest czescia wszystkich serwerów aplikacji, które mozna klastrowac. Zapewnia ona wspóldzielony, globalny dla klastra, kontekst JNDI, poprzez który klient moze wyszukiwac oraz rejestrowac obiekty. Zarejestrowane wpisy sa w HA-JNDI replikowane w calym klastrze, dzieki czemu obiekt moze byc wyszukany pomimo awarii wezla, w którym zostal zarejestrowany. Mechanizm ten dziala niezaleznie od zwyklego lokalnego kontekstu JNDI kazdego serwera, w którym rejestrowane sa wszystkie obiekty domowe komponentów na nim dzialajacych.

Zdalny klient, wyszukujacy obiekt poprzez usluge HA-JNDI dzialajaca na wezle N, moze napotkac sytuacje, w których komponent byl zarejestrowany poprzez:

- HA-JNDI,

- lokalna usluge JNDI wezla N,
- lokalna, ale dzialajaca na innym wezle niz N, usluge JNDI.



Rys. 8 Wyszukiwanie komponentów EJB poprzez HA-JNDI.

W zaleznosci od sposobu zarejestrowania komponentu operacja wyszukania komponentu (1) *lookup()* przebiega nastepujaco - Rys. 8:

- Jezeli wpis jest dostepny w globalnym drzewie JNDI zostanie on zwrócony (1.1).
- Jesli brak wpisu w drzewie globalnym, zapytanie jest delegowane do lokalnego serwisu JNDI, który zwraca odpowiedz jesli jest ona osiagalna (1.2).
- W przypadku braku wpisów zarówno w globalnym jak i lokalnym drzewie JNDI, odpytywane sa inne wezly w klastrze (1.3 oraz 1.3'). Jesli ich lokalne serwisy JNDI posiadaja wlasciwe wpisy zwracany jest wybrany element sposród zbioru uzyskanych odpowiedzi (1.4 i 1.4').
- Jezeli zaden z lokalnych serwisów JNDI nie ma odpowiedniego wpisu zglaszany jest wyjatek *NameNotFoundException*.

Kazde wyszukanie, realizowane poprzez HA-JNDI, dla którego brak wlasciwego wpisu w rejestrze HA-JNDI jest delegowane do lokalnego rejestru JNDI. Poniewaz komponenty EJB sa rejestrowane w lokalnym rejestrze, beda one zawsze wyszukiwane z wykorzystaniem mechanizmu delegacji. Z faktu tego plyna dwa wazne wnioski:

- Jezeli komponent jest dostepny jedynie na kilku wezlach w klastrze, istnieje wysokie prawdopodobienstwo, ze odpytywany wezel nie jest dla wyszukiwanego komponentu lokalnym, co spowoduje przekierowanie zapytania do innych wezlów. Konsekwencja tej operacji jest duzo dluzszy czas oczekiwania na odpowiedz, niz w przypadku dostepnosci wpisu w lokalnym rejestrze. W celu poprawy wydajnosci, raz uzyskane wyniki wyszukania JNDI nalezy umieszczac w pamieci podrecznej

klienta. Mozna to uzyskac poprzez wykorzystanie wzorca projektowego lokalizatora serwisów (ang. service locator).

- Wykorzystanie lokalnego drzewa JNDI w celu znalezienia obiektu zarejestrowanego w HA-JNDI spowoduje wygenerowanie wyjatku *NameNotFoundException*, poniewaz HA-JNDI ma wlasne drzewo JNDI.

Ostatnim elementem, który rózni HA-JNDI od zwyklego JNDI jest sposób laczenia klienta z usluga. W przypadku zwyklego JNDI, klient posiada w konfiguracji zapis zawierajacy nazwe hosta oraz port, na którym dziala usluga nazewnicza. Podobne informacje potrzebne sa dla polaczenia z usluga HA-JNDI. Problemem pozostaje jednak sprawa wyboru hosta oraz sytuacja jego niedostepnosci, np. z powodu awarii. Jednym z mozliwych rozwiazan jest podawanie w konfiguracji zamiast jednego wpisu, calej listy dostepnych adresów, pod którymi znajduja sie uslugi JNDI. W czasie inicjalizacji nastapi próba polaczenia z kolejnymi serwisami z listy, az do uzyskania polaczenia. Rozwiazanie to, w przypadku silnie zmiennych konfiguracji klastra, moze byc trudne do wlasciwego zarzadzania – czyli dostarczania klientom aktualnej listy wezlów. W konsekwencji serwis HA-JNDI zostal rozszerzony o mechanizm automatycznego wykrywania (ang. auto discovery). Najprostsza usluga automatycznego wykrywania dziala w sposób nastepujacy. Jesli wpis konfiguracji klienta zawierajacy nazwe i port serwisu pozostanie pusty albo zaden z serwisów z listy nie jest dostepny, podjeta zostanie próba wyszukania uslugi HA-JNDI poprzez zapytanie grupowe. Rozglaszanie grupowe umozliwia, w sieciach lokalnych lub rozleglych sieciach WAN, dzialanie klientów bez zadnej konfiguracji poczatkowej [LB03].

2.4.3 Optymalizacja zapisu i dostepu do danych

Relacyjne bazy danych sa podstawowymi zródlami danych aplikacji komponentowych. Ich wydajnosc warunkuje bezposrednio wydajnosc dzialania aplikacji zwiazana z operacjami na danych (wyszukiwanie, dostep, dodawanie, uaktualnianie, usuwanie). Serwery aplikacji posiadaja mechanizmy umozliwiajace zmniejszenie obciazenia bazy danych. Najbardziej znanym i najczesciej stosowanym jest mechanizm pul polaczen, polegajacy na utrzymywaniu przez serwer otwartych polaczen z baza danych. Polaczenia takie sa udostepniane komponentom, co redukuje czas potrzebny na kazdorazowe inicjowanie polaczenia [Jon03]. Inne mechanizmy opieraja sie glównie na podrecznej – czesto wielopoziomowej – pamieci dla danych oraz zapytan. Pamiec taka pozwala na istotna redukcje ilosc powtarzalnych operacji odczytu. W przypadku bardzo duzej liczby zapytan

oraz aplikacji intensywnie zapisujacych dane mechanizmy te nie sa wystarczajace. Zwiekszenie wydajnosci systemu wymaga zwiekszenia wydajnosci bazy danych.

Klasteryzacja bazy danych

Najbardziej popularna forma zwiekszania wydajnosci bazy danych jest klasteryzacja. Autor [Mul02] wymienia dwa podstawowe modele klasteryzacji baz danych: architekture wspólnego systemu plików (ang. shared disk architecture) oraz architekture bez współdzielen (ang. shared nothing architecture); ich wlasnosci zbiera Tabela 3.

Architektura „shared disk”	Architektura „shared nothing”
Wymaga bardzo wydajnych konfiguracji sprzetowych	Dziala na mniej wydajnych, tanszych konfiguracjach sprzetowych
Zapewnia wysoka dostepnosc	Praktycznie nieograniczona skalowalnosc
Dane nie musza byc partycjonowane	Wymaga partycjonowania danych
Szybkie dopasowanie do zmieniajacego sie obciazenia	Zwiekszanie wydajnosc wymaga rozbudowy konfiguracji sprzetowej

Tabela 3 Wlasnosci podstawowych modeli klasteryzacji baz danych.

Nalezy podkreslic, ze bez wzgledu na zastosowany model, z punktu widzenia wykorzystujacego baze danych systemu komponentowego, pozostaje ona pojedynczym, spójnym zródlem danych. Kwestia optymalizacji wewnetrznych rozwiazan jest domena szczególowych badan specjalistów z obszaru baz danych i lezy poza zakresem pracy; informacje z tego zakresu czytelnik moze znalezc w [Amb04][Ree01][JLV+03].

Najwieksza róznica obu podejsc jest, poza wewnetrzna konstrukcja, kwestia partycjonowania danych, która przybliza kolejny punkt.

Partycjonowanie danych

Partycjonowanie danych jest pojeciem wywodzacym sie z obszaru baz danych, definiowanym w [Mor02] nastepujaco:

> **Partycjonowanie danych** (ang. data partitioning) polega na automatycznym rozpraszaniu danych (pochodzacych z jednej lub wielu relacji) na wielu dyskach, znajdujacych sie w tym samym lub wielu wezlach (komputerach) sieci.

Partycjonowanie danych z wielu relacji polega na zapisywaniu tabel bazy danych w róznych lokalizacjach i pozostaje stosunkowo nieskomplikowanym procesem, polegajacym na przypisaniu okreslonym tabelom róznej lokalizacji fizycznej. Partycjonowanie danych z

jednej relacji bazuje na podziale danych zawartych w jednej tabeli bazy danych. Zarówno zapis danych jak i pózniejszy do nich dostep, wymaga wykorzystania wydajnych algorytmów partycjonujacych, które decyduja o fizycznej lokalizacji konkretnego rekordu. Autor [WB03] wymienia trzy podstawowe algorytmy: partycjonowanie bazujace na wartosci, partycjonowanie mieszajace (ang. hashing) oraz partycjonowanie hybrydowe.

Zyski z partycjonowania danych sa bardzo wymierne:

- kosztowne operacje wejscia/wyjscia moga byc wykonywane równolegle,
- równowazone jest obciazenie dysków,
- polecenia SQL moga byc wykonywane równolegle,
- wzrasta szybkosc tworzenia kopii zapasowych bazy danych oraz ich odtwarzania po awarii.

Z zastosowaniem partycjonowania wiaza sie równiez aspekty negatywne:

- potrzeba podzialu zapytan i agregacji wyników,
- wieksze wymagania sprzetowe,
- dodatkowe narzuty zwiazane z synchronizacja transakcji, przeliczaniem indeksów,
- dodatkowe narzuty czasowe zwiazane z procesem partycjonowania, szczególnie dotkliwe w przypadku blednego podzialu danych.

Mechanizmy partycjonowania danych w systemach komponentowych mozna podzielic na dwie kategorie. Pierwsza obejmuje wykorzystanie partycjonowania danych wewnetrznie przez wbudowane mechanizmy bazy danych. Partycjonowanie pozostaje wtedy niewidoczne dla serwera aplikacji oraz komponentów i jest mechanizmem calkowicie ukrytym w wewnetrznej strukturze silnika bazy danych. Jego stosowanie i uzyskany wzrost wydajnosci jest silnie uwarunkowany konkretna implementacja bazy danych. Druga kategoria obejmuje podzial danych pomiedzy niezalezne instancje baz danych, z których kazda jest rozróznialna i indywidualnie dostepna dla serwera aplikacji. Do partycjonowania i agregacji danych wykorzystywane sa specjalizowane komponenty, a baza danych nie musi implementowac zaawansowanych mechanizmów zwiazanych z tymi operacjami.

Partycjonowanie danych, jako sposób na zwiekszenie wydajnosci oraz uzyskanie rozwiazania skalowalnego, jest zalecane szczególnie w przypadku aplikacji intensywnie zapisujacych dane [Pru05] [Dor02] [Nau03].

Partycjonowanie danych, realizowane wewnatrz bazy danych, wymaga najczesciej wykorzystania drogich, komercyjnych silników bazy danych. Skutkuje to silnym przywiazaniem tworzonego systemu do konkretnej bazy danych. Alternatywa, szczególnie dla popularnych rozwiazan opartych na licencjach otwartych (ang. open-source), jest wykorzystanie biblioteki C-JDBC. C-JDBC jest oprogramowaniem posredniczacym, umozliwiajacym, dowolnej aplikacji napisanej w jezyku Java, przezroczysty dostep do klastra baz danych. Dostep ten jest realizowany poprzez protokól JDBC, nie wymaga modyfikowania aplikacji oraz wsparcia dla mechanizmów klasteryzacji i partycjonowania po stronie baz danych [CMP+05]. C-JDBC, bazuje na koncepcji jednej wirtualnej bazy danych, dzialajacej w oparciu o mechanizmy replikacji i partycjonowania danych z róznych relacji. Fakt ten skutkuje uzyskaniem wlasnosci równowazenia obciazenia, wysoka dostepnoscia, mozliwoscia monitorowania i analizy wydajnosci zapytan SQL oraz wsparciem dla heterogenicznych baz danych. Wada biblioteki jest brak mozliwosci partycjonowania danych nalezacych do jednej tabeli bazy danych.

Wykorzystywane w serwerach aplikacji pule polaczen sa konfigurowane dla kazdej instancji bazy danych niezaleznie. Dlatego partycjonowanie danych realizowane przez komponenty wymaga wykorzystania, w ramach jednego serwera aplikacji, niezaleznych pul polaczen do wszystkich instancji baz danych uczestniczacych w partycjonowaniu. Rozwiazaniem tego problemu jest mechanizm zbiorczej puli polaczen (ang. multi-pool), czyli puli polaczen zarzadzajacej polaczeniami do kilku instancji baz danych [BEA04]. Niestety, jest on niedostepny w wiekszosci implementacji serwerów aplikacji.

Mechanizmy klasteryzacji bazy danych oraz partycjonowania danych realizowane przez komponenty zostaly wykorzystane przez autora w celu zwiekszenia wydajnosci tworzonego systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych.

2.4.4 Wykorzystanie brokera komunikatów

Kolejnym mechanizmem, który mozna stosowac w celu poprawienia wydajnosci systemu jest broker komunikatów. Z jego wykorzystaniem zwiazane sa trzy podstawowe wlasnosci: asynchronicznosc i kolejkowanie operacji oraz równowazenie obciazenia.

Wywolania asynchroniczne stosuje sie z reguly w stosunku do tych operacji, które ze swej natury wykonuja sie dlugo. Moze to byc generowanie zlozonego raportu, albo wywolanie operacji na systemie zewnetrznym, np. systemie obslugi kart platniczych czy rezerwacji

biletów. Wywolania takich operacji nie powinny byc realizowane w trybie synchronicznym, bowiem zasoby, zajete przez wywolujacy operacje watek, pozostaja niewykorzystane i niedostepne dla innych, przez caly (dlugi) czas wykonywania operacji.

Wlasnosc kolejkowania operacji rozszerza koncepcje wywolania asynchronicznego o mozliwosc „oczekiwania” operacji na wykonanie. W przypadku niedostepnosci lub przeciazenia podsystemu majacego wykonac zadana operacje, nie jest ona odrzucana ale oczekuje w kolejce, az jej wykonanie bedzie mozliwe. Mechanizm kolejek pozwala na zachowania parametrów wydajnosciowych systemu przy jego przeciazeniu, nie zwieksza jednak ilosc operacji (komunikatów) jakie jest w stanie przetworzyc system odbiorczy. Zbyt wysokie obciazenie utrzymujace sie w dluzszym okresie czasu, spowoduje osiagniecie maksymalnej dlugosci kolejki i odmowe przyjecia komunikatu, co jest widziane przez klienta jako awaria systemu [Nau03]. Mechanizm kolejkowania sprawdza sie bardzo dobrze przy chwilowych przeciazeniach systemu (pikach w obciazeniu). Umozliwia bowiem przetworzenie nadmiarowych wywolan w pózniejszym czasie, podczas gdy bez jego zastosowania, nadmiarowe wywolania zostalyby odrzucone od razu.

Dzieki zastosowaniu brokera komunikatów, latwo uzyskuje sie wlasnosc równowazenia obciazenia pomiedzy odbiorcami wiadomosci. Odbiorcy informuja broker o gotowosci do przetworzenia kolejnego komunikatu niezaleznie od siebie. Bardziej wydajny odbiorca bedzie to robil czesciej, a tym samym bedzie czesciej otrzymywal od brokera komunikat do przetworzenia, niz odbiorca dysponujacy mniejsza wydajnoscia.

W architekturze J2EE broker komunikatów jest zgodnym ze specyfikacja JMS (ang. Java Messaging Service) podsystemem kolejkowym, który umozliwia wspólprace róznych elementów systemu poprzez wymiane komunikatów [SSJ+02]. Obsluguje on dwa tryby wymiany komunikatów: punkt-punkt (ang. point-point) oraz wydawca-prenumerator (ang. publish-subscribe). W pierwszym trybie, komunikat nadany przez nadawce trafia wylacznie do jednego odbiorcy. W drugim, nadany komunikat jest odczytywany przez wszystkich zainteresowanych odbiorców. Nadawca komunikatów moze byc dowolny obiekt Java dzialajacy w ramach serwera aplikacji albo poza nim. Zgodnie ze specyfikacja JMS odbiorca komunikatu moze byc dowolny obiekt implementujacy interfejs *MessageListener*. Specyfikacja EJB wyróznia specjalne, sterowane zdarzeniami komponenty (ang. Message Driven Bean), które powinny byc odbiorcami komunikatów; zostaly one juz opisane wczesniej. Wykorzystanie komponentów sterowanych zdarzeniami umozliwia kontenerowi komponentów zarzadzanie ich iloscia oraz wykorzystywanymi przez nie zasobami [SUN03c].

Praktycznie kazdy serwer aplikacji jest obecnie dostarczany razem z brokerem komunikatów, istnieje tez pokazna ilosc niezaleznych implementacji, które dzieki JMS latwo wykorzystac w dowolnej aplikacji J2EE. Wykorzystanie brokera komunikatów przy zapisywaniu duzej ilosci danych w bazie danych jest bardzo obiecujaca koncepcja. Zostala ona przez autora wykorzystana w dalszej czesci pracy – punkt 4.3 – jako alternatywa dla klasycznego podejscia bazujacego na bezposredniej komunikacji z baza danych.

2.5 Podsumowanie

Przedstawiony w tym rozdziale przeglad technologii lezacych u podstaw pracy obejmowal w pierwszej czesci przeglad rozwiazan z zakresu monitorowania systemów rozproszonych, które sa zródlem danych dla tworzonego systemu zbierania i przechowywania danych. Autor wymienia szereg róznych technologicznie srodowisk monitorujacych oraz blizej przedstawia kilka wybranych. Analiza modeli informacyjnych, modeli danych oraz modeli dystrybucji danych jest podstawa dla zaproponowania w rozdziale 3 wymagan, ogólnej architektury, modelu danych oraz uniwersalnych interfejsów dostepu do danych.

W dalszej czesci rozdzialu przedstawiono aktualne srodowiska dla budowy systemów komponentowych. Sposród dostepnych rozwiazan, z uwagi na otwartosc oraz duza liczbe implementacji, , autor wybral platforme J2EE jako bazowa dla konstrukcji sytemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych. W kolejnym punkcie przyblizona zostala koncepcja warstwowej budowy aplikacji J2EE, podstawowe typy komponentów platformy oraz ich wlasnosci. Kolejny punkt zostal poswiecony technikom zwiekszania wydajnosci systemów komponentowych. Zdefiniowane zostaly pojecia wydajnosci oraz zwiekszania wydajnosci systemu komponentowego. Przedstawiono mozliwosci w zakresie optymalizacji dzialania komponentów, kontenera i serwera aplikacji, klasteryzacji serwera aplikacji, wykorzystania brokera komunikatów oraz mechanizmy optymalizacji zapisu danych do bazy danych, oparte na zwielokrotnianiu instancji bazy danych oraz partycjonowaniu danych. Mozliwosci te zostaly wykorzystywane do budowy, przedstawionych w rozdziale 4, rozszerzen modelu systemu, tworzonych w celu uzyskania wlasnosci skalowalnosci oraz odpowiedniej wydajnosci systemu.

3 Model systemu zbierania i przechowywania danych pochodzacych z monitorowania

"It is not the strongest of species that survive, nor the most intelligent, but the one most adaptable to change"[1]
- Charles Darwin

Celem niniejszego rozdzialu jest wykazanie prawdziwosci czesci tezy mówiacej o mozliwosci budowy komponentowego systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych, którego dzialanie jest adaptowalne do zmian zarówno rodzaju danych pochodzacych z konkretnego zasobu jak i trybu ich zbierania. Rozdzial przedstawia zagadnienia konstrukcji Systemu Zbierania i Przechowywania Danych pochodzacych z monitorowania systemów rozproszonych (SZiPD). Otwiera go lista wlasnosci, jakimi tworzony system winien sie charakteryzowac. Nastepnie przedstawiono analize mozliwych rozwiazan z zakresu architektury systemu. Szczególowo przedstawiono model informacyjny systemu, zaproponowano wlasciwy dla niego ogólny, obiektowy model danych oraz uniwersalne interfejs dostepu do danych. Kolejne punkty przyblizaja kwestie miejsca systemu w ogólnym modelu monitoringu. Rozdzial zamyka propozycja komponentowej implementacji modelu oraz weryfikacja jego funkcjonalnosci w swietle postawionych w tezie wymagan. Kwestie wydajnosci i skalowalnosci systemu zostaly przedstawione w rozdziale 4.

3.1 Wymagania dla SZiPD

Z przeprowadzonej w punkcie 2.1 analizy systemów monitorujacych oraz postawionych we wstepie dysertacji wymagan dla wspólczesnych systemów tego typu wynika szczególowa lista wlasnosci, jakie powinny charakteryzowac system zbierajacy i przechowujacy dane pochodzace z monitorowania. Obejmuje ona nastepujace postulaty:

- Róznorodnosc monitorowanych zasobów. System ma gromadzic dane pochodzace z monitorowania róznych zasobów. Powinien umozliwiac zapis i przechowywanie danych z róznych domen podlegajacych monitorowaniu np. sieci komputerowych, jednostek obliczeniowych, pamieci masowych itp.

[1] Gatunkiem, który przezyje, nie jest najinteligentniejszy ani najsilniejszy, ale ten, który najlepiej przystosowuje sie do zmian.

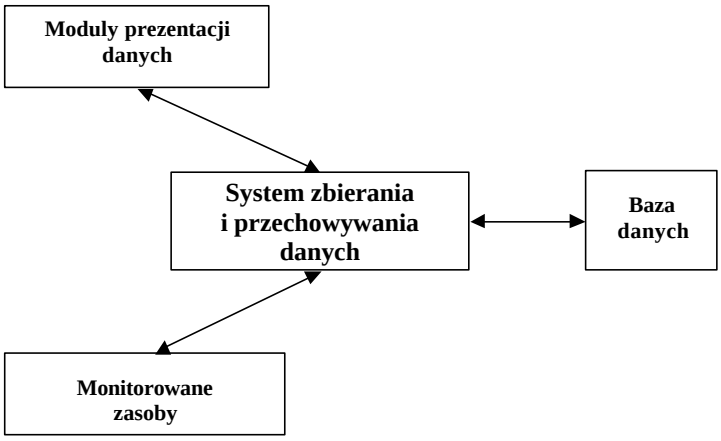
- Dynamiczne definiowanie zasobów. System musi umozliwiac rejestrowanie i usuwanie zasobów podlegajacych monitorowaniu w czasie swojego dzialania.
- Dynamiczne definiowanie atrybutów. System musi umozliwiac rejestrowanie atrybutów podlegajacych monitorowaniu na danym zasobie oraz latwe rozszerzanie tej listy o nowe atrybuty, które zasób udostepnil juz w czasie dzialania systemu.
- Obsluga atrybutów zlozonych. Analizowane systemy monitorujace wykorzystywaly logiczne grupowanie atrybutów w wieksze struktury (np. atrybut ‘statystyka procesora’ moze skladac sie z kilku atrybutów: czas bezczynnosci, srednie obciazenie itp.).
- Obsluga atrybutów wielowartosciowych. System musi obslugiwac sytuacje, w których wartosc atrybutu wyrazona jest kilkoma wartosciami (ilosc ta moze byc rózna w czasie). Przykladem takiego wielowartosciowego atrybutu moze byc atrybut „zalogowani uzytkownicy systemu operacyjnego”, jego wartosc sklada sie z kilku wartosci bedacych nazwami aktualnie zalogowanych uzytkowników.
- Obsluga róznych typów danych. System powinien uwzgledniac mozliwie szerokie spektrum prostych typów danych wystepujacych w systemach monitorujacych.
- Rózna czestotliwosc zapisu. Konstrukcja systemu musi uwzgledniac mozliwa, rózna czestotliwosc zapisu danych pochodzacych z jednego zasobu. System powinien umozliwiac niezalezny zapis wartosci poszczególnych atrybutów oraz grupowanie danych.
- Jednolity interfejs zapisu danych. Pomimo mozliwej duzej zmiennosci atrybutów, udostepnianych przez zasób do monitorowania, system powinien udostepniac jednolity interfejs dla zapisu danych o zasobach, atrybutach i ich strukturze oraz samych wartosci.
- Jednolity interfejs dostepu do danych. Pomimo róznorodnosci zasobów podlegajacych monitorowaniu, system powinien udostepniac jednolity interfejs dostepu zarówno do informacji o zasobach, atrybutach i ich strukturze jak i do zgromadzonych wartosci. Interfejs powinien umozliwiac selekcje i filtrowanie zwracanych wartosci.

- Rózne tryby monitorowania. System powinien wspierac trzy podstawowe tryby monitorowania atrybutów: raportowanie (ang. push), odpytywanie (ang. pull) oraz sledzenie zmian wartosci (ang. tracing).
- Wydajnosc dzialania i skalowalna budowa. Kluczowym zagadnieniem przy konstrukcji SZiPD jest opracowanie rozwiazania skalowalnego, a wiec zdolnego do zapewnienia wymaganych parametrów jakosciowych, pomimo wzrostu ilosci wykonywanych na systemie operacji oraz objetosci danych.

Najwiekszym wyzwaniem dla konstrukcji systemu opartego o powyzsza liste wymagan jest konstrukcja uniwersalnego modelu danych oraz uniwersalnych interfejsów zapisu i dostepu do danych. Nowatorskie jest zalozenie takiej konstrukcji systemu, która umozliwi wykorzystanie jako zródlo danych róznych zasobów oraz mozliwosc dynamicznego definiowania monitorowanych atrybutów. Oba problemy zostaly rozwiazane poprzez zaproponowanie modelu informacyjnego oraz odpowiedniego dla niego modelu danych bazujacego na koncepcji meta-danych dla systemów monitorujacych. Podstawa dla dyskusji tych zagadnien jest przedstawienie ogólnej architektury sytemu.

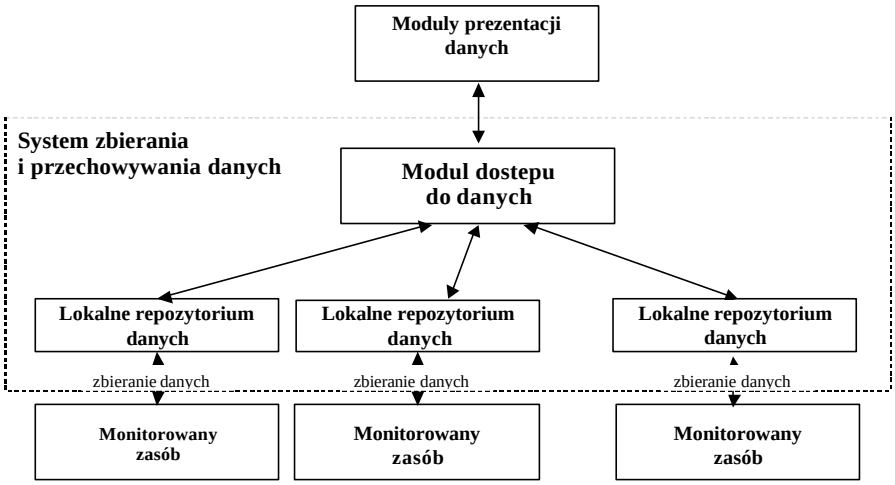
3.2 Ogólna architektura SZiPD

Klasyczna architektura systemów zbierania i przechowywania danych pochodzacych z monitorowania jest przedstawiona na Rys. 9. Wyróznia ona elementy wspólpracujace z systemem, a wiec zasoby, które udostepniaja podlegajace monitorowaniu wartosci, moduly prezentacji danych oraz baze danych. System zbierania i przechowywania posredniczy w dostepie do bazy danych zarówno przy wykonywaniu operacji zapisu jak i odczytu danych.



Rys. 9 SZiPD z centralna baza danych.

Alternatywne rozwiazanie, bazujace na mozliwosci zapisu danych w elementach posredniczacych w dostepie do zasobów, obrazuje Rys. 10. Monitorowane zasoby posiadaja dodatkowo dedykowane repozytoria danych które sa wykorzystywane przez modul udostepniajacy dane dla prezentacji.



Rys. 10 SZiPD z rozproszonym repozytorium danych.

Wybierajac jedno z powyzszych rozwiazan nalezy rozwazyc sposób zapamietywania danych. Dane mozna bowiem przechowywac w postaci zródlowej (ang. native data, raw data) czyli takiej w jakiej otrzymano je ze monitorowanego zasobu, a nastepnie konstruowac system o funkcjonalnosci hurtowni danych, który polaczy takie heterogeniczne repozytoria. Alternatywa jest opracowanie wspólnego dla wszystkich danych formatu oraz odpowiednich mechanizmów konwersji do tego formatu. O ile w przypadku operacji zapisu oba rozwiazania wydaja sie byc równie dobre (nawet ze wskazaniem na format zródlowy), o tyle stopien skomplikowania operacji wyszukania i odczytu wskazuje na rozwiazanie ze wspólnym formatem. Przykladowo: operacja wyszukania – musi byc zapisana w formacie ogólnym, a nastepnie odwzorowana na formaty specyficzne. Równoczesnie wynik jej dzialania, zebrany z kilku róznych zródel, powinien dawac homogeniczny zbiór odpowiedzi, tylko taki bowiem mozna rozsadnie prezentowac. Proces ten jest bardzo skomplikowany, zawiera bowiem operacje sprowadzania do wspólnego formatu oraz dodatkowo szereg innych zlozonych transformacji, czego unika sie konwertujac dane do wspólnego formatu przed zapisem.

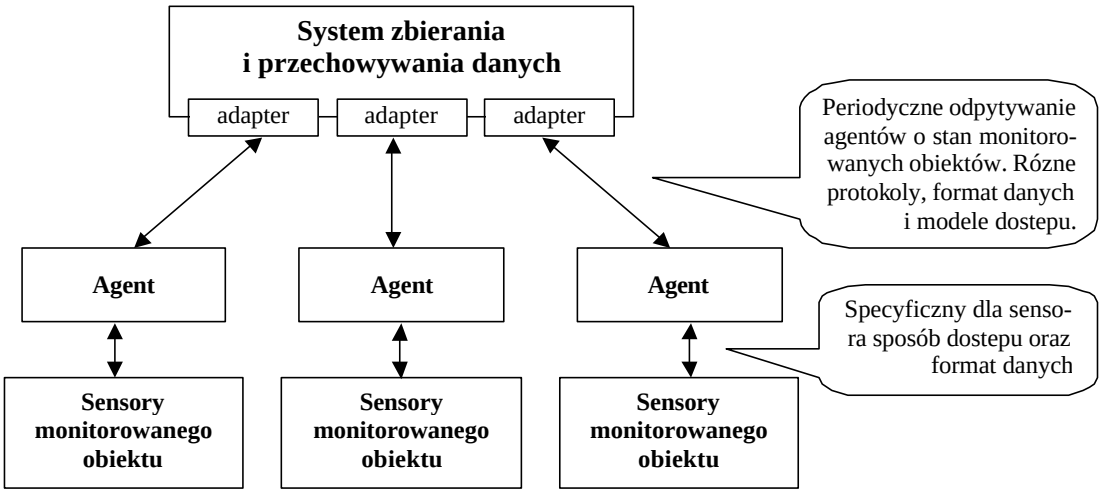
Oba rozwiazania wymagaja wprowadzenia, pomiedzy system zbierania i przechowywania danych, a zasób, elementu posredniczacego. Element ten moze byc implementowany jako czesc systemu bazowego lub byc traktowany jako niezalezny element zewnetrzny. Jego funkcjonalnosc moze byc ograniczona jedynie do konwersji danych lub moze obejmowac obsluge lokalnego repozytorium danych, konwersje zapytan oraz odpowiedzi. Element taki

bardzo czesto wystepuje we wspólczesnych systemach monitorujacych i jest nazywany **agentem monitorujacym**.

W kolejnych punktach przedstawiono koncepcje laczenia systemu SZiPD z agentami monitorujacymi o róznej funkcjonalnosci.

3.2.1 Wykorzystanie agenta zewnetrznego systemu monitorujacego

Rozwiazanie problemu zbierania przez SZiPD danych z róznych zasobów mozna uzyskac wykorzystujac agentów zewnetrznych systemów monitorujacych - Rys. 11. Kwestia komunikacji agent – monitorowany zasób jest dzieki temu rozwiazywana przez twórców zewnetrznego systemu, a tworzony SZiPD zyskuje mozliwosc pozyskiwania danych z bardzo szerokiego spektrum zasobów dostepnych przez istniejacych agentów. Podejscie takie jest bardzo atrakcyjne, napotyka jednak na powazne problemy zwiazane z integracja róznorodnych technologicznie agentów z tworzonym system. Najprostszy, pokazany na Rys. 11, przypadek zaklada jedynie odpytywanie agentów o stan zasobu oraz obsluge róznych formatów danych. Kompletne rozwiazanie wymagaloby od systemu obslugi szeregu mechanizmów, specyficznych dla konkretnych implementacji agentów, zwiazanych z raportowaniem i powiadomieniami. Innym problemem jest calkowity brak kontroli nad aktywnoscia agentów, iloscia i czestotliwoscia przesylanych przez nich danych. Fakt ten moze w znaczacy sposób wplywac na wydajnosc, a nawet stabilnosc dzialania tworzonego systemu.



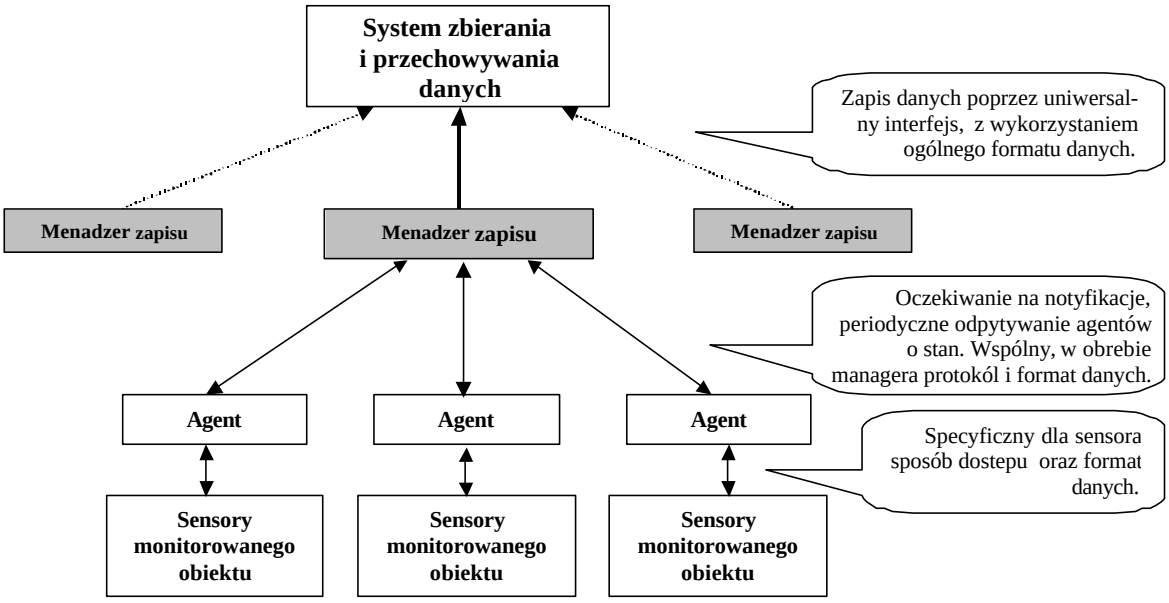
Rys. 11 Dostep do zasobów poprzez agentów zewnetrznych systemów monitorujacych.

Duza trudnoscia przy implementacji rozwazanego podejscia jest sposób dynamicznego instalowania w systemie adapterów dla konkretnego agenta oraz stworzenie mechanizmu przydzielania wlasciwego adaptera do dolaczanego do systemu agenta. W celu rozwiazania

powyzszych problemów zaproponowany zostal, przedstawiony w kolejnym punkcie, modul menadzera zapisu.

3.2.2 Modul menadzera zapisu

Modul menadzera zapisu jest elementem posredniczacym pomiedzy agentem a SZiPD. Menadzer zapisu komunikuje sie z agentem przez specyficzny dla agenta protokól, realizuje konwersje danych do uniwersalnego formatu oraz zapisuje je w SZiPD poprzez uniwersalny interfejs – Rys. 12.



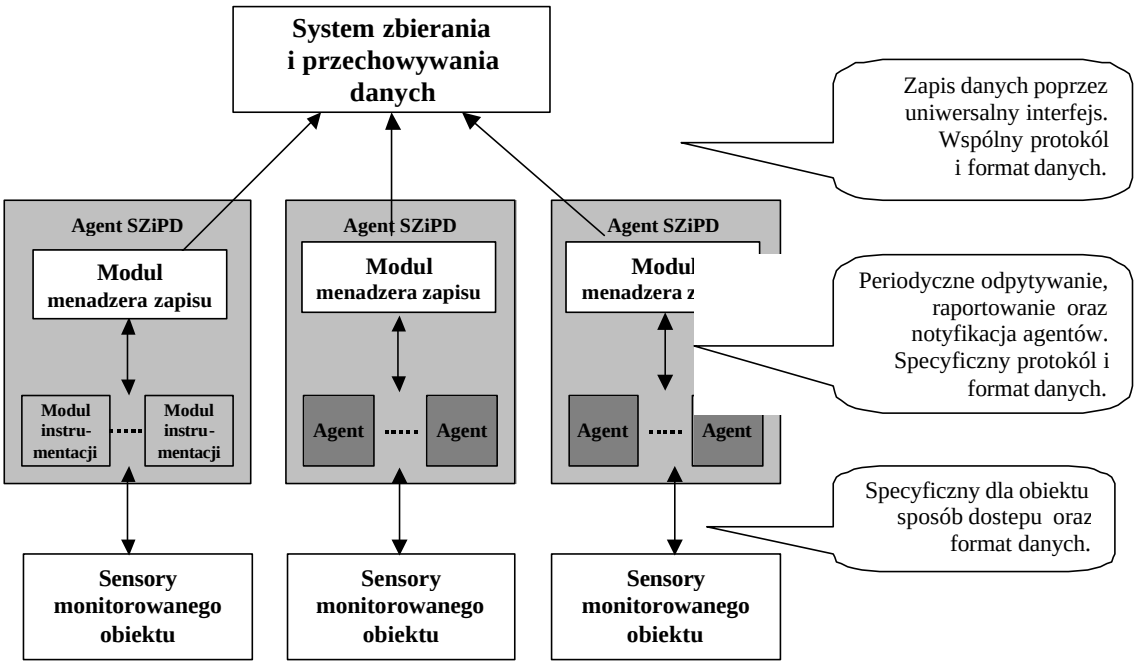
Rys. 12 Dostep do zasobu z wykorzystaniem menadzera zapisu.

Menadzer zapisu jest ulepszonym adapterem, posiada wieksza autonomie oraz jest tworzony dla grupy agentów dzialajacych w oparciu o ten sam protokól komunikacyjny oraz zbiór danych. Wymaga to jego implementowania dla agentów róznych systemów monitorujacych. Menadzer zapisu jest niezalezna od SZiPD aplikacja. Podejscie takie rozwiazuje, sygnalizowany w opisywanej wczesniej koncepcji, problem dynamicznego instalowania adapterów oraz umozliwia instalacje menadzera w poblizu agenta/grupy agentów. Menadzer zapisu posredniczy w dostepie do homogenicznych agentów, a pozyskane dane konwertuje do wspólnego formatu i zapisuje przez uniwersalny interfejs, zdefiniowany w dalszej czesci pracy. Proponowane rozwiazanie upraszcza tworzenie samego SZiPD, który jedynie udostepnia uniwersalny interfejs zapisu. Rola menadzera jest oczekiwanie na notyfikacje badz odpytywanie agentów, filtrowanie danych oraz przesylanie ich do systemu. Umozliwia to optymalizacje komunikacji system – menadzer poprzez wprowadzenie mechanizmu grupowania danych przed przekazaniem do systemu.

Wada rozwiazania jest fakt, ze menadzer zapisu jest dodatkowa aplikacja, co komplikuje proces uruchamiania oraz zarzadzanie systemem jako caloscia, szczególnie w przypadku wiekszej liczby menadzerów. Problemem jest równiez sytuacja, w której system ma zbierac dane z zasobu, dla którego nie ma agenta monitorujacego oraz sposób dostosowania ilosci instancji menadzerów zapisu do ilosci agentów w celu unikniecia efektu ‘waskiego gardla’. Rozwiazanie powyzszych kwestii mozna uzyskac poprzez zastosowanie agenta SZiPD.

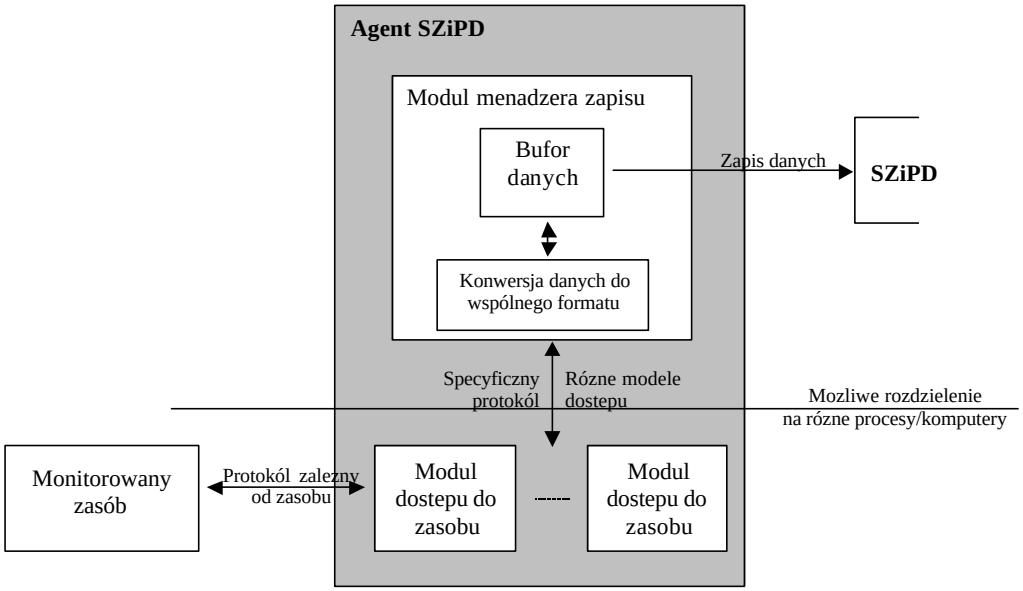
3.2.3 Agent SZiPD

Agent SZiPD sklada sie z dwóch podsystemów: modulu menadzera zapisu oraz modulów dostepu do zasobu. Rola modulu menadzera zapisu jest zgodna z przedstawiona w poprzednim punkcie, moduly dostepu do zasobów moga byc agentami systemu monitorujacego badz implementacja modulu instrumentujacego monitorowany zasób – Rys. 13.



Rys. 13 Dostep do zasobu z wykorzystaniem agenta SZiPD.

Koncepcja agenta SZiPD jest bardziej elastyczna niz prezentowana w poprzednim punkcie koncepcja menadzera zapisu. Umozliwia wykorzystanie agentów monitorujacych pochodzacych z zewnetrznych systemów monitorowania lub wlasnych modulów instrumentacji, w przypadku braku agenta lub jego niewystarczajacej funkcjonalnosci. Proponowane rozwiazanie zaklada mozliwosc dzialania modulu dostepu do zasobu oraz modulu menadzera zapisu w róznych procesach a nawet na róznych komputerach – Rys. 14.



Rys. 14 Struktura agenta SZiPD.

Modul dostepu do zasobu moze byc zrealizowany jako agent systemu monitorujacego lub jako wlasny modul instrumentacji zasobów. W przypadku agenta systemu monitorujacego nie mozna wplywac na udostepniony przez niego interfejs. Problem integracji jest rozwiazywany przez odpowiednia implementacje modulu menadzera zapisu, dedykowanego dla tego agenta. Modul ten zapewnia konwersje danych do wspólnej reprezentacji; moze równiez uzupelniac funkcjonalnosc zewnetrznego agenta poprzez udostepnienie elementów zwiazanych z zarzadzaniem procesem monitorowania. W szczególnosci, dla najprostszych i najbardziej popularnych agentów, które jedynie udostepniaja dane, modul ten moze umozliwic sterowanie czestoscia próbkowania, filtrowanie danych na podstawie wartosci progowych oraz sledzenie wartosci.

Podstawowym zadaniem modulu menadzera zapisu jest zapisywanie danych w SZiPD. Dane sa przekazywane do modulu menadzera zapisu w jednym z trzech trybów monitorowania, w formacie specyficznym dla modulu dostepu do zasobu (agent, modul instrumentacji). Nastepnie, dane podlegaja konwersji do wspólnego formatu oraz sa przekazywane do wewnetrznego bufora danych. Wewnetrzny bufor gromadzi dane przed ich wyslaniem do SZiPD. Bufor jest periodycznie opróżniany, a dane zapisywane w systemie poprzez uniwersalny interfejs zapisu. Zastosowanie bufora umozliwia transport danych w wiekszych porcjach i moze w znaczy sposób obnizyc zaburzenia wprowadzane do monitorowanego systemu poprzez współdzielenie medium komunikacyjnego oraz zapewnic wsparcie dla monitorowania posredniego (ang. off-line). Poprzez odpowiednie ustalanie objetosci bufora oraz czestosci przekazywania danych, w sposób plynny mozna balansowac pomiedzy malymi

opóznieniami, okupionymi wiekszym obciazeniem medium komunikacyjnego, a duzymi opóznieniami, dzieki którym w wybranym okresie nie bedzie zadnych dodatkowych obciazen medium.

Poniewaz celem niniejszej pracy jest budowa SZiPD, a nie agenta monitorujacego, dalsze kwestie zwiazane z konstrukcja agenta nie beda w pracy szczególowo dyskutowane. Autor ograniczy sie jedynie do implementacji agenta niezbednej dla przeprowadzenia testów wydajnosciowych SZiPD.

3.2.4 Wnioski

Przeprowadzone w punkcie rozwazania prowadza do postawienia nastepujacych wniosków:

- budowa SZiPD wymaga zdefiniowania ogólnego modelu danych dla systemów monitorujacych; model taki jest niezbedny dla zapewnienia mozliwosci zapisywania w SZiPD danych pochodzacych z róznych istniejacych systemów monitorujacych,
- niezbedne jest stworzenie uniwersalnych interfejsów dostepu do systemu; interfejsy takie zapewnia mozliwosc zapisywania i pobierania danych zgodnych z ogólnym modelem danych; podejscie takie nie wplywa na ograniczenie funkcjonalnosci zwiazanej z przyjmowaniem danych w róznych trybach monitorowania,
- poprzez implementacje agentów SZiPD mozliwa jest integracja SZiPD z agentami istniejacych systemów monitorujacych, oraz pozyskiwanie danych bezposrednio z zasobów,
- dolaczanie do systemu nowych implementacji agentów SZiPD nie wymaga przebudowy systemu.

Kluczowym dla konstrukcji SZiPD pozostaje okreslenie modelu informacyjnego, wlasciwego dla niego modelu danych oraz uniwersalnych interfejsów dostepu do danych. Problematyce tej poswiecony jest kolejny punkt.

3.3 Model informacyjny SZiPD

Dobrze zaprojektowany model informacyjny jest kluczem do zapewnienia uniwersalnosci SZiPD. Uniwersalnosc jest rozumiana jako zdolnosc do zapisywania oraz udostepniania mozliwie najszerszego spektrum danych o rozmaitych typach i strukturze, jakie sa wykorzystywane w systemach monitorujacych.

Kazda dana dostepna w systemie informatycznym ma sens jedynie jesli znane jest jej znaczenie, innymi slowy, jesli istnieje pewna skojarzona z nia informacja okreslajaca zarówno typ jak i semantyke danej. Dla przykladu dana: ‘65’ moze posiadac wiele znaczen, moze byc kodem ASCII litery A, wiekiem pacjenta, liczba procesorów w systemie, oraz kluczem glównym rekordu bazy danych. Model informacyjny, w szczególnosci dla informacji pochodzacych z monitorowania, jest kompletny jesli obejmuje nie tylko same wartosci monitorowanych parametrów ale równiez zwiazane z nimi meta-dane, bedace kontekstem dla interpretacji tych wartosci. Specyfika informacji pochodzacych z monitorowania polega na tym, ze meta-dane moga byc dostepne poza systemem, w którym zostaly wygenerowane, tak wiec kontekst ten musi byc znany i opisany w ramach tworzonego modelu informacyjnego.

Przedstawione w punkcie 2.1, wybrane rozwiazania z zakresu monitorowania systemów rozproszonych byly podstawa do analizy wykorzystywanych przez te systemy modeli informacyjnych oraz modeli danych z róznych dziedzin informatyki: obliczenia, sieci komputerowe, infrastruktura sprzetowa. Niniejszy punkt zawiera definicje pojec specyficznych dla monitorowania systemów rozproszonych, takich jak zasób oraz atrybut, tworzacych model informacyjny SZiPD. Model ten jest podstawa do zaproponowania w dalszej czesci tego rozdzialu ogólnego obiektowego modelu danych pochodzacych z monitorowania systemów rozproszonych.

3.3.1 Zasób monitorowany

Zasób monitorowany (ang. resource) jest dowolnym bytem udostepniajacym pewien zbiór danych charakteryzujacy jego stan w danej chwili czasu. Dane o stanie musza byc dostepne przez interfejs programistyczny, dostepny albo bezposrednio w zasobie, albo poprzez system informatyczny majacy polaczenie z zasobem. Zasób musi posiadac unikalna w ramach lokalnej instancji systemu nazwe, która bedzie go w sposób jednoznaczny identyfikowac.

Przykladami zasobów moga byc: komputer PC o adresie IP 149.156.97.82, pacjent o numerze PESEL 760825032350, aplikacja identyfikowana przez numer procesu oraz IP komputera na którym dziala.

Przestrzen nazw zasobów

Z uwagi na róznorodnosc zasobów z jakimi mozna miec do czynienia w obrebie jednego systemu, nalezy wyróznic pojecie przestrzeni nazw (ang. kind, ang. namespace).

Przestrzenie nazw grupuja w sposób rozlaczny podobne zasoby. Kazdy zasób nalezy do jednej i tylko jednej przestrzeni nazw. Wprowadza to duze uporzadkowanie oraz ulatwia wyszukiwanie zasobów poprzez dostep do listy zasobów konkretnego rodzaju np. komputerów, routerów czy pacjentów. Dodatkowo dzieki wprowadzeniu przestrzeni nazw zyskuje sie kontekst dla interpretacji nazw atrybutów. Wprowadzanie bardziej zaawansowanych rozwiazan grupujacych nie wydaje sie uzasadnione i nie jest praktykowane w zadnym ze znanych autorowi systemów monitorowania.

3.3.2 Atrybut monitorowanego zasobu

> **Atrybut** (ang. *attribute*), cecha charakteryzujaca obiekt rzeczywisty lub abstrakcje, element opisu takiego obiektu. Na przyklad atrybutem zmiennej jest jej typ, jednym z atrybutów pliku jest jego wielkosc, atrybutem procesora jest liczba operacji wykonywanych w ciagu sekundy, w standardzie katalogowym X.500 atrybutem identyfikowanej encji moze byc pole “hobby”, natomiast atrybutem okregu jest jego promien.[Plo99]

W systemach monitorujacych atrybutem nazywa sie podlegajaca monitorowaniu ceche monitorowanego zasobu. Nazywana jest ona równiez parametrem albo metryka. Zawsze jednak oznacza cos wiecej niz jedynie wartosc badz zbiór wartosci. W najprostszych systemach monitorujacych atrybut posiada jedynie nazwa, bardziej rozbudowane systemy wprowadzaja dodatkowe wlasnosci jak opis oraz typ danych, jaki przyjmuja wartosci. Analiza róznych systemów monitorujacych wykazala istnienie róznych rodzajów atrybutów: prostych, strukturalnych, wielowartosciowych. Zostaly one przedstawione ponizej.

Atrybuty proste

Najprostszym rodzajem atrybutów udostepnianych przez zasób sa atrybuty, których typem danych jest typ prosty. Z uwagi na duza ilosc róznorodnych typów danych, dostepnych w róznych jezykach programowania, proponuje sie na potrzeby modelu trzy podstawowe typy proste: liczbowy (dla liczb naturalnych), zmiennoprzecinkowy (dla liczb rzeczywistych) oraz napis dla danych tekstowych. Nalezy podkreslic, ze praktycznie wszystkie proste typy danych sa latwo konwertowane do trzech wyróznionych. Pewna nadmiarowosc wynikajaca z wprowadzenia typu liczbowego (moze byc reprezentowana przez zmiennoprzecinkowy) wynika z jego wielkiej popularnosci, oraz trudnosci w porównywaniu typów zmiennoprzecinkowych wynikajacej z róznic w reprezentacji na róznych platformach. Proste porównanie z typami danych dostepnych w jezykach programowania rodzi watpliwosc odnosnie kompletnosci rozwiazania, nie zawiera ono bowiem tablicowego typu binarnego,

potrzebnego np. zapisu strumieni danych [Wir89]. Typ taki nie jest jednak potrzebny poniewaz obiekty w systemach monitorujacych pojawiaja sie zamiennie ze strukturami, maja bowiem za zadanie wylacznie opakowanie danych prostych. Powinny byc zatem reprezentowane jako struktury, gdyz tylko wtedy ich poszczególne pola beda mogly byc porównywane – serializacja nie jest tutaj dobrym rozwiazaniem i nie powinna byc stosowana. Inne dane o charakterze binarnym (np. obraz) nie podlegajace porównywaniu i wyszukiwaniu, nie wystepuja w systemach monitorujacych praktycznie wcale, w zwiazku z tym nie ma potrzeby wprowadzania dla nich specjalnego typu. Ich ewentualne uzycie jest mozliwe poprzez konwersje bajtów do standardu ASCII i zapis jako typ ‘napis’.

Nazwa	**Typ**	**Wartosc**
Nazwisko pacjenta	Napis	Kowalski
Ilosc procesorów	Liczba	3
Dostepna przepustowosc [kbps]	Liczba	36600
Zajeta pamiec RAM [MB]	Liczba	128

Tabela 4 Przykladowe dane atrybutów prostych.

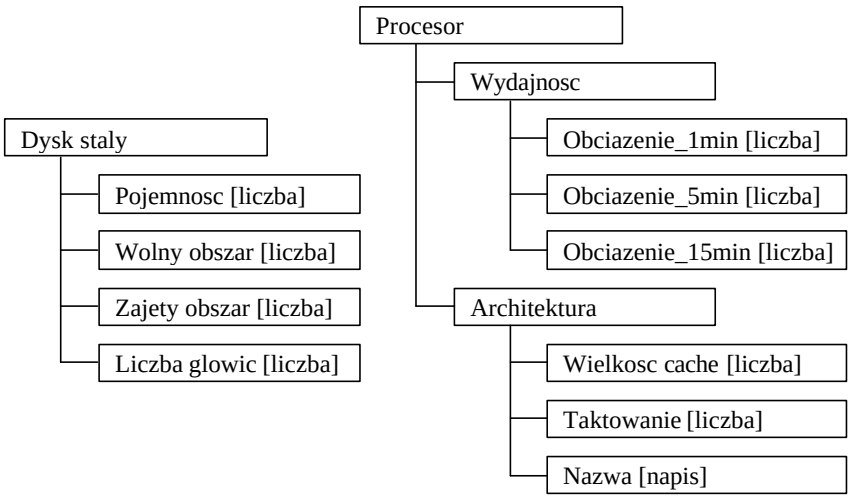
Atrybuty proste sprawdzaja sie doskonale w wiekszosci sytuacji, nie obejmuja jednak wszystkich danych udostepnianych przez bardziej zaawansowane systemy. Wyrózniono dodatkowo dwa rodzaje atrybutów: strukturalne i wielowartosciowe.

Atrybuty strukturalne

> Najbardziej ogólna metoda tworzenia **typów zlozonych** jest laczenie w typ zlozony elementów o dowolnych, byc moze zlozonych typach. Przykladami z matematyki sa liczby zespolone zlozone z dwóch liczb rzeczywistych, czy wspólrzedne punktów zlozone z dwóch lub wiecej liczb rzeczywistych w zaleznosci od wymiaru przestrzeni. Przykladem z przetwarzania danych jest opis osoby za pomoca kilku istotnych cech charakterystycznych jak imie, nazwisko, data urodzenia, plec. [CDS+88]

Wystepowanie w systemach monitorujacych typów zlozonych wymaga wprowadzenia atrybutu strukturalnego. Atrybut strukturalny jest kolekcja innych atrybutów, dzieki czemu umozliwia tworzenie zlozonych struktur z atrybutów prostych oraz innych atrybutów strukturalnych. Powstaje w ten sposób graf atrybutów. Poniewaz tworzenie atrybutów strukturalnych jest glównie dzialaniem porzadkujacym, majacym na celu lepsza organizacje atrybutów prostych, nie dopuszcza sie cyklicznosci tak powstalego grafu. Zalozenia te prowadza do powstania struktury drzewiastej, w której lisciach znajduja sie atrybuty proste,

wezlami zas sa atrybuty strukturalne; nie ma przy tym ograniczenia odnosnie wysokosci drzewa – Rys. 15.



Rys. 15 Przykladowe atrybuty strukturalne.

Cecha charakterystyczna atrybutów strukturalnych jest brak wartosci – grupuja one inne atrybuty, a jedynie bedace liscmi, atrybuty proste, posiadaja wartosci. Konstrukcja taka jest zgodna ze sposobem opisu atrybutów w analizowanych systemach monitorujacych.

Tworzenie atrybutów strukturalnych jest dzialaniem porzadkujacym – stosunkowo latwo jest przeprowadzic transformacje takich struktur na zbiór atrybutów prostych poprzez odpowiednie modyfikowanie nazw atrybutów prostych. Tabela 5 obrazuje taka transformacje wykonana dla atrybutu „Dysk staly” z Rys. 15 przy uzyciu separatora „_%_”.

Nazwa	**Typ**	**Wartosc**
Dysk staly_%_pojemnosc	Liczba	30 000 000
Dysk staly_%_wolny obszar	Liczba	10 000 000
Dysk staly_%_zajety obszar	Liczba	20 000 000
Dysk staly_%_ilosc glowic	Liczba	5

Tabela 5 Transformacja atrybutów strukturalnych do prostych.

Pierwotnie model danych zawieral takie uproszczenie, stosowane z powodzeniem np. w SNMP oraz w bibliotece PDH opisanych w punktach 2.1.1 oraz 2.1.2. Problematyczne okazalo sie jednak laczenie tak uproszczonych atrybutów z atrybutami wielowartosciowymi. SNMP nie dopuszcza bowiem zagniezdzania atrybutów w atrybutach wielowartosciowych, które moga byc jedynie liscmi w drzewie MIB. Nie jest zatem mozliwe posiadanie struktury np. opisujacej procesor jako typu strukturalnego i wielowartosciowego zarazem. W tworzonym modelu wlasnosc tego typu jest pozadana, wiec przytoczone uproszczenie nie jest stosowane.

Atrybuty wielowartosciowe

Atrybuty wielowartosciowe to takie, które posiadaja w danym momencie kilka wartosci. Ceche ta moga posiadac zarówno atrybuty proste jak i strukturalne. Atrybuty wielowartosciowe mozna podzielic na dwie grupy: indeksowane i nieindeksowane. Wartosci wielowartosciowego atrybutu nieindeksowanego tworza rodzaj zbioru – nie jest istotna ich kolejnosc, ale mozliwe sa powtórzenia. Przykladowo: imiona dzieci pacjenta, nazwy aktualnie zalogowanych uzytkowników, itp. – Tabela 6. Wartosci wielowartosciowego atrybutu indeksowanego tworza liste; sa w jakis sposób uporzadkowane innymi wartosciami; a wiec indeksowane. Przykladowo, adresy interfejsów sieciowych routera indeksowane sa nazwa tego interfejsu, obciazenie procesora w maszynie wieloprocesorowej jest indeksowane identyfikatorem procesora itp.

Nazwa	**Typ**	**Wartosc**
Nazwy zalogowanych uzytkowników	Napis	„radzisz”, „piter”, „sloik”, „kbalos”
Nazwy aktywnych procesów	Napis	„java”, „Xsystem”, „bash”
Numery zajetych portów	Liczba	80, 443, 8080, 24

Tabela 6 Przykladowe dane atrybutów wielowartosciowych.

Przyjete rozróznienie atrybutów wielowartosciowych zostalo jednak uproszczone tak, ze atrybut wielowartosciowy jest zawsze rozumiany jako atrybut wielowartosciowy nieindeksowany. Wynika to z dwóch powodów; po pierwsze mozliwe jest istnienie bardzo zlozonych indeksów (skladajacych sie np. z kilku wartosci), a próba bezposredniego opisania takich przypadków bardzo komplikuje model. Po drugie wielowartosciowy atrybut indeksowany mozna zapisac jako nieindeksowany z wykorzystaniem atrybutu strukturalnego. Atrybut strukturalny moze swoich elementach skladowych zawierac dowolne atrybuty, które beda interpretowane jak indeksy. Rozwiazanie to ma równiez te zalete, ze pozwala na istnienie kilku róznych niezaleznych od siebie indeksów.

V[ind1]->val1 ⇔ Struktura(‘index’->ind1, ‘wartosc’->val1)
V[ind2]->val2 Struktura(‘index’->ind2, ‘wartosc’->val2)

Przykladowo wielowartosciowy atrybut indeksowany nazwami interfejsów routera zawierajacy ich adresy mozna zapisac jako wielowartosciowy atrybut nieindeksowany:

V[“int1”]->”149.156.97.99” ⇔ Struktura(‘index’->“int1”, ‘wartosc’->”149.156.97.99”)
V[“int2”]->”149.156.96.1” Struktura(‘index’->“int2”, ‘wartosc’->”149.156.96.1”)

Podejscie takie pozwala na ujednolicenie obslugi atrybutów wielowartosciowych bez utraty ogólnosci rozwiazania. Poniewaz tworzony model danych ma byc modelem obiektowym, zostal on przedstawiony w dalszej czesci pracy przy uzyciu notacji UML (ang. Unified Modeling Language). Autorowi nie jest obca notacja BNF [Nau60][Wir82], przy pomocy której mozna opisac przedstawione wyzej rozwazania, jednak podejscie takie mogloby blednie wskazywac na próbe stworzenia specyficznej gramatyki, co nie jest celowe.

Znacznik czasowy

Wykorzystywane do tej pory pojecie wartosci atrybutu ograniczalo sie do jednej wartosci oraz ewentualnie zbioru wartosci w przypadku atrybutów wielowartosciowych. Poniewaz system monitorujacy udostepnia wartosci zmienne w czasie, dlatego z wartoscia atrybutu nalezy zawsze kojarzyc znacznik czasowy opowiadajacy chwili czasu w którym zostaly zmierzone. W tym kontekscie pojecie wartosci atrybutu odnosi sie do zmierzonej w konkretnej chwili czasu wartosci, a wartosc atrybutu wielowartosciowego jest rozumiana jako zbiór wartosci zmierzonych w tym samym momencie czasu.

Nadawanie znaczników czasowych powinno byc realizowane w chwili pomiaru, czyli najlepiej w zasobie, a para wartosc i znacznik powinna byc juz zawsze widziana lacznie. Czesto jednak interfejs zasobu udostepnia jedynie wartosci, wtedy przyporzadkowanie znacznika czasowego musi zostac dokonane przez agenta lub modul instrumentacji. Bez wzgledu jednak na miejsce, w którym to przyporzadkowanie ma miejsce, otwartym problemem pozostaje kwestia synchronizacji zegarów elementów systemu. Jest ona dyskutowana w szeregu pracach [DB01a][DB01b][DB01c][Lam78] i nie bedzie szerzej rozwazana.

Dodatkowe dane opisujace atrybut

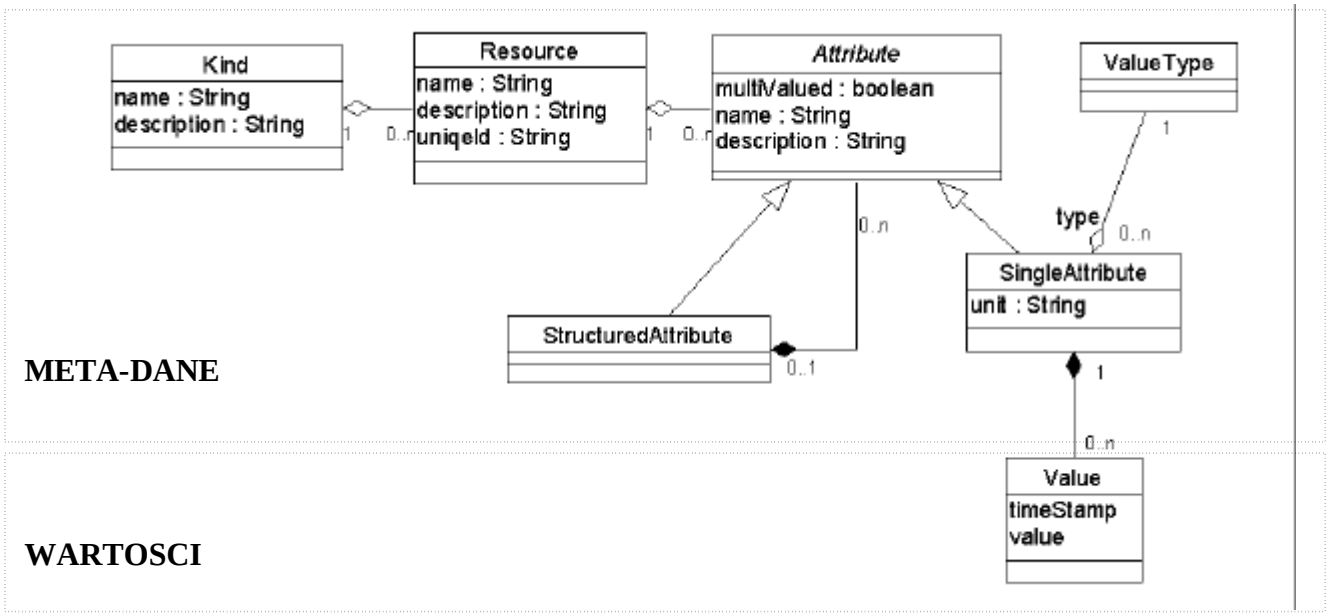
Przeglad systemów prezentacji danych pochodzacych z monitorowania wykazal potrzebe rozszerzenia skladowych atrybutu: nazwy i typu danych o dwa dodatkowe pola zawierajace informacje semantyczna:

- opis – dokladny opis atrybutu, rozszerzajacy informacje zawarta w polu nazwa,
- jednostke – pole szczególnie uzyteczne dla danych liczbowych, znacznie ulatwiajace prezentacje.

Jednostka jest charakterystyczna wylacznie dla atrybutów posiadajacych wartosci, a wiec atrybutów prostych. Pole opis dotyczy wszystkich rodzajów atrybutów.

3.3.3 Ogólny obiektowy model danych

Opisane w punkcie 3.3.2, elementy domeny systemu stanowia podstawe dla zaproponowania ogólnego obiektowego modelu danych pochodzacych z monitorowania systemów rozproszonych – Rys. 16. Obiektowosc modelu danych jest jednym z zalozen postawionych w rozwinieciu tezy pracy. Pozostale, przyjete w pracy zalozenia, w tym komponentowe podejscie do tworzenia systemu, w zaden sposób nie wplywaja na postac zaproponowanego modelu danych.



Rys. 16 Ogólny model danych pochodzacych z monitorowania.

Ogólny obiektowy model danych obejmuje dwie grupy danych: meta-dane czyli informacje opisujace dane (monitorowane zasoby oraz ich atrybuty) oraz wartosci atrybutów, czyli konkretne, zmienne w czasie, wartosc monitorowanych atrybutów.

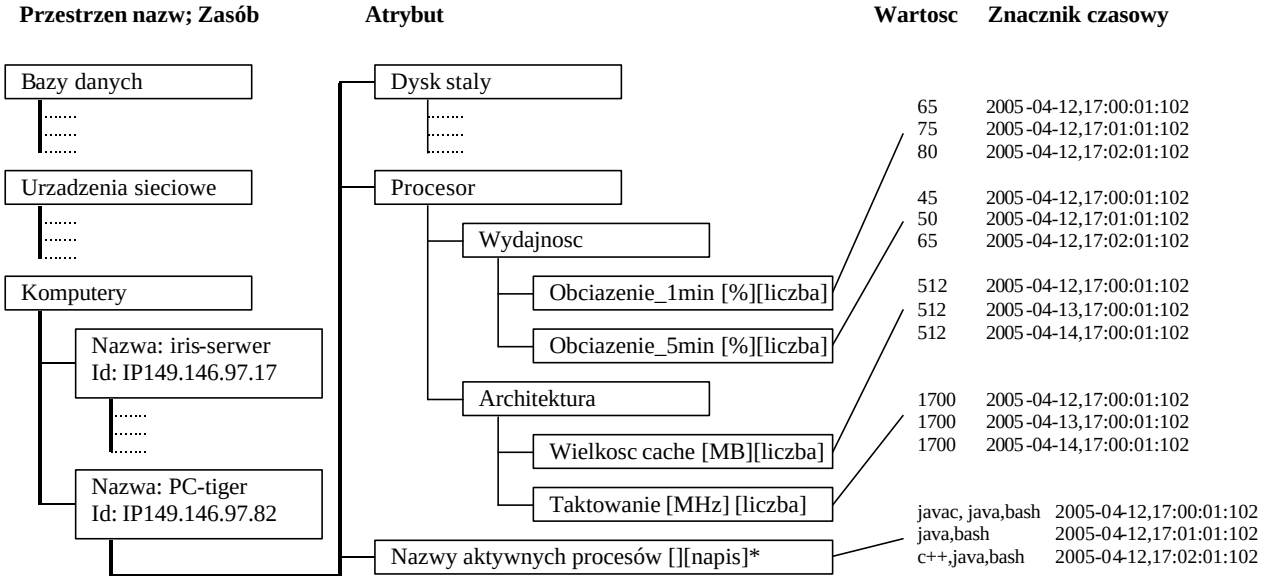
Meta-dane obejmuja:

- Przestrzen nazw (ang. kind) — grupuje podobne zasoby, czyli te opisywane przez podobne atrybuty (np. komputery, urzadzenia sieciowe). Jest opisywana przez nazwe i opis.
- Zasób (ang. resource) — reprezentuje monitorowany obiekt, jest opisywany unikalnym identyfikatorem zasobu, nazwa zasobu, opisem, posiada tez referencje do przestrzeni nazw, do której nalezy,
- Atrybut (ang. attribute) — reprezentuje monitorowany parametr. Zgodnie z przedstawionym wczesniej opisem, atrybuty moga sie zagniezdzac tworzac

drzewiaste struktury. Atrybut jest zatem klasa abstrakcyjna posiadajaca dwie podklasy: *StructuredAttribute* sluzaca do budowy tych struktur oraz *SimpleAttribute* reprezentujaca atrybut prosty. Atrybut moze opisywac zarówno dane skalarne jak i wielowartosciowe, wlasnosc ta moze byc regulowana przez odpowiednie ustawienie pola *multiValued*. Atrybut jest opisywany przez nazwe, opis oraz referencje zasobu, którego jest elementem. Atrybut prosty posiada dodatkowo jednostke oraz typ danych (klasa *ValueType*).

Wartosc atrybutu

Wartosc atrybutu jest reprezentowana przy pomocy klasy *Value*, klasa ta zawiera trzy atrybuty: wartosc, znacznik czasowy oraz referencje do atrybutu prostego.



Rys. 17 Przykladowe meta-dane i wartosci atrybutów.

Wartosc – jest zmierzona w danym momencie wartoscia monitorowanego atrybutu; zgodnie z przeprowadzona wczesniej argumentacja moze byc wyrazona przy pomocy jednego z ponizszych typów:

- napis – dla danych tekstowych,
- liczba – dla danych o typie naturalnym,
- liczba zmiennoprzecinkowa – dla danych o typie zmiennoprzecinkowym.

Znacznik czasowy – jest nadanym w chwili pomiaru znacznikiem okreslajacym moment dokonania pomiaru.

Referencja do atrybutu – laczy wartosc z calym kontekstem niezbednym do interpretacji jej znaczenia, a wiec informacja o atrybucie i zasobie, z jakiego ta wartosc pochodzi.

Przykladowe wartosci meta-danych oraz wartosci atrybutów przedstawia Rys. 17.

3.3.4 Wnioski

W punkcie 3.3 zostaly zdefiniowane pojecia z zakresu domeny systemów monitorujacych: zasób, atrybut, atrybut strukturalny, atrybut wielowartosciowy, przestrzen nazw. Definicja tych pojec byla podstawa dla zaproponowania ogólnego obiektowego modelu danych pochodzacych z monitorowania systemów rozproszonych. Stworzony model zostal logicznie podzielony na dwie czesci: meta-dane i wartosci (dane).

Model obiektowy jest zawsze jedynie pewnym przyblizeniem i uproszczeniem modelowanej domeny, obejmujacym jej wybrane aspekty [Lar02]. Proponowany ogólny, obiektowy model danych pochodzacych z monitorowania systemów rozproszonych zostal utworzony w oparciu o analize mozliwie duzej liczby systemów monitorujacych. Przedstawione rozwazania sa odpowiednie dla tych systemów, a model danych pozwala na opisanie udostepnianych przez nie danych. Nie mozna jednak kategorycznie stwierdzic, ze nie istnieje, badz nie zostanie stworzony system monitorujacy, dla którego proponowany ogólny model bedzie niewystarczajacy.

3.4 SZiPD w warstwowym modelu monitoringu

W niniejszym punkcie przedstawiono sposób wspólpracy elementów architektury systemów monitorujacych z SZiPD. Przedstawione mechanizmy sa podstawa dla zdefiniowania w kolejnym punkcie uniwersalnych interfejsów dostepu do SZiPD, których stworzenie zaklada rozwiniecie tezy pracy.

SZiPD powinien wpisywac sie w ogólny, warstwowy model monitoringu. Zaproponowany przez autora [Lau01] model monitoringu wyrózniajacy trzy podstawowe warstwy, zostal rozszerzony o dwie dodatkowe warstwy: instrumentacji i dostepu; taki wlasnie, pieciowarstwowy model jest rozwazany. Tabela 7 pokazuje do jakich warstw naleza poszczególne elementy architektury systemu oraz wyróznia elementy bedace integralnymi czesciami SZiPD.

Warstwa w modelu monitoringu	Element architektury	Projekt i implementacja	Przeplyw monitorowanych danych
Warstwa prezentacji	Graficzny klient		
Warstwa dostepu do danych	**Uniwersalny interfejs dostepu**	**X**	
Warstwa przechowywania danych	**System zbierania i przechowywania danych**	**X**	
Warstwa zbierania danych	**Uniwersalny interfejs zapisu danych**	**X**	
Warstwa instrumentacji	Agent monitorujacy		
Zasób	Specyficzny interfejs dostepu		
	Zasób		

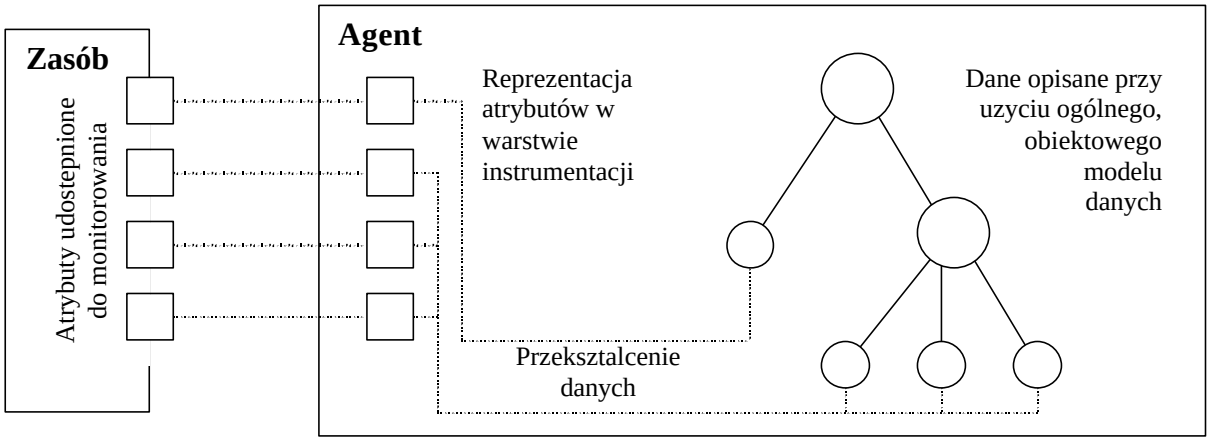
Tabela 7 SZiPD w warstwowym modelu monitoringu.

Kolejne punkty przedstawiaja funkcjonalnosc poszczególnych warstw z uwzglednieniem wyborów z zakresu architektury systemu, dokonanych w punkcie 3.2, przedstawionego w punkcie 3.3 modelu danych oraz przeplywu i zmiany struktury danych podlegajacych monitorowaniu.

3.4.1 Warstwa instrumentacji

Glównym zadaniem warstwy instrumentacji jest pozyskiwanie danych z zasobu. Wykorzystywany jest do tego specyficzny interfejs zasobu, poprzez który dane sa dostepne oraz agent monitorujacy, który dane te potrafi pozyskac. Mozliwe rozwiazania w tym obszarze byly dyskutowane w punkcie 3.2. Uzywane tam pojecie wspólnego formatu danych zostalo doprecyzowane w punkcie 3.3.3 i zostalo okreslone mianem wspólnego obiektowego modelu danych.

Zadaniem agenta monitorujacego jest stworzenie zgodnej ze wspólnym obiektowym modelem danych struktury danych: meta-danych oraz samych wartosci. Dane pobierane z zasobu musza byc przeksztalcone do ogólnego obiektowego modelu danych. Przeksztalcenie to, w najprostszym przypadku, moze byc zwyklym odwzorowaniem, najczesciej liniowej, struktury danych zasobu na zbiór atrybutów prostych. Pelne wykorzystanie mozliwosci systemu zaklada stworzenie na tym poziomie dowolnie zlozonych struktur drzewiastych opartych na atrybutach strukturalnych – Rys. 18. Moga one wynikac wprost ze struktury danych monitorowanego zasobu, moga tez byc stworzone na podstawie dodatkowych informacji posiadanych przez agenta. Stworzenie wlasciwej, drzewiastej struktury meta-danych, oddajacej logiczna strukture monitorowanych danych pozostaje w gestii agenta, który ma w tym zakresie pelna dowolnosc.



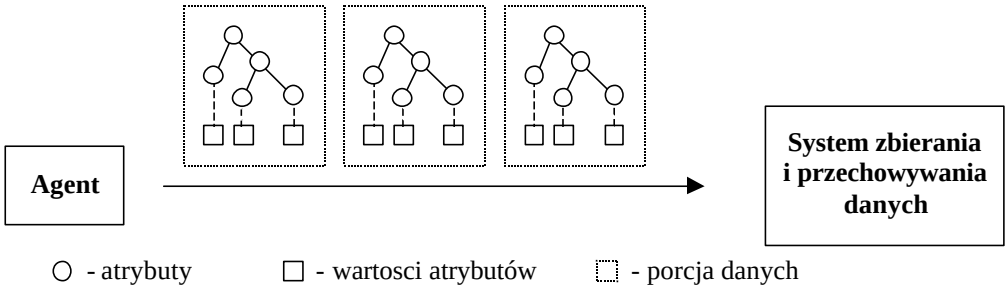
Rys. 18 Konwersja danych wewnatrz agenta.

Tworzenie atrybutów prostych wymaga od agenta wiedzy o typie wartosci podlegajacych monitorowaniu oraz ich krotnosci. Informacja ta, jesli jest dostepna, moze byc pobrana z zasobu, najczesciej jednak bedzie pochodzic z innego zródla np. pliku konfiguracyjnego agenta. Zadaniem agenta jest równiez konwersja wartosci monitorowanych parametrów do okreslonych w atrybutach typów. Powyzsze dzialania zapewniaja wyzszej warstwie (zbierania danych), mozliwosc operowania na takiej samej dla wszystkich zasobów i atrybutów strukturze danych, opartej na ogólnym, obiektowym modelu danych.

3.4.2 Warstwa zbierania danych

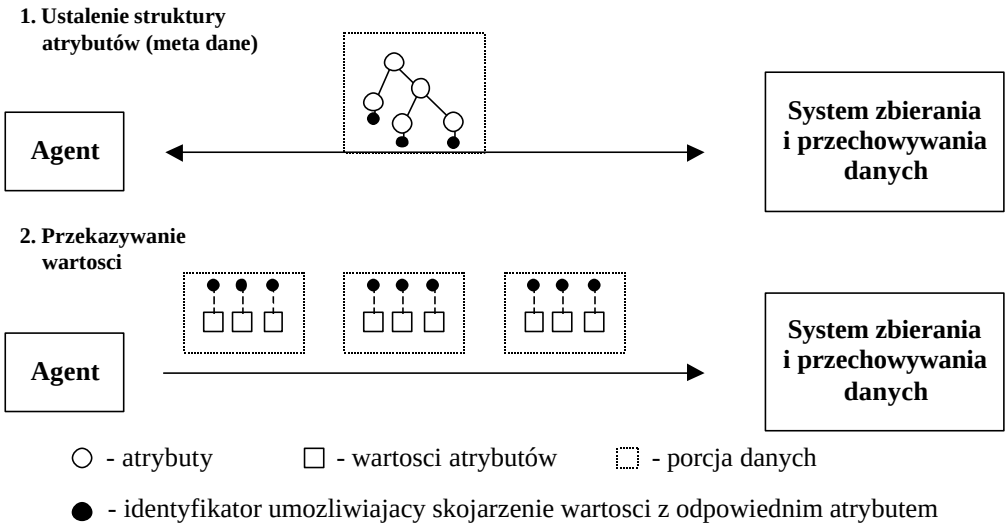
W warstwie zbierania danych miesci sie calosc dzialan zwiazanych z przekazywaniem danych od agentów do systemu. Obejmuje ona czesciowo samego agenta, mechanizmy i protokól transportu oraz interfejs systemu odpowiedzialny za przyjmowanie danych. Przedstawiona w punkcie 3.2 argumentacja pozwala wykorzystac uniwersalny interfejs zapisu danych oraz przeniesc aktywnosc zwiazana z jego wykorzystaniem na agenta. Agent monitorujacy dziala w dwóch warstwach: instrumentacji oraz kolekcji danych. Jego zadaniem jest pozyskanie danych z zasobu, ich przygotowanie oraz przeslanie do systemu. Proces przeslania danych moze byc realizowany na dwa sposoby.

Pierwszy sposób polega na dystrybucji konkretnej zmierzonej wartosci wraz z cala informacja potrzebna do jej interpretacji (meta-dane), a wiec cala struktura atrybutów – Rys. 19. Rozwiazanie to jest stosowane przewaznie do integracji róznych systemów monitorujacych. Ma ono jednak zasadnicze wady w zakresie wydajnosci: przesylane jest duzo nadmiarowej informacji, która trzeba z jednaj strony przygotowac, a z drugiej przetworzyc, co, poza obciazeniem medium komunikacyjnego, powoduje zwiekszenie obciazenia jednostek przetwarzajacych.



Rys. 19 Przekazywanie wartosci wraz z cala struktura atrybutów.

Drugie podejscie zaklada przeslanie w pierwszym kroku informacji opisujacej meta-dane, a nastepnie wykorzystanie identyfikatorów, nadanych przez system atrybutom prostym. Zmierzone wartosci sa po stronie agenta dodatkowo wyposazane we wlasciwy identyfikator atrybutu, dzieki któremu po przeslaniu do systemu moga byc skojarzone i poprawnie przyporzadkowane odpowiednim atrybutom – Rys. 20.



Rys. 20 Rozdzielne przekazywanie danych pochodzacych z monitorowania.

Przedstawione podejscia sa zasadniczo rózne, zarówno na poziomie koncepcji, implementacji jak i mozliwosci zdefiniowania ogólnego interfejsu zapisu danych. SZiPD zostal oparty na koncepcji rozdzielnego przekazywania danych i meta-danych, glównie z nastepujacych powodów:

- Uzyskiwane dzieki rozdzielnemu przekazywaniu danych, zmniejszenie obciazenia kanalów transmisji danych zalezy od proporcji wielkosci danych i meta-danych. Zaproponowany dla SZiPD ogólny obiektowy model danych jest stosunkowo zlozony. Wielkosc meta-danych w stosunku do monitorowanych wartosci jest bardzo duza, szczególnie w przypadku monitorowania wartosci liczbowych.

Rezygnacja z przekazywania meta-danych z kazda wartoscia powoduje istotne obnizenie wielkosci przesylanych danych.

- Przesylanie meta-danych razem z kazda wartoscia nie wnosi do tworzonego systemu zadnej istotnej dodatkowej funkcjonalnosci.
- Faza ustalania struktury atrybutów pozwala wykryc niespójnosc struktury posiadanej przez agenta z juz istniejaca w systemie, a opisujaca ten sam zasób. Chodzi o sytuacje, w której atrybut o tej samej nazwie ma w systemie np. inny typ. Problem taki moze byc od razu zgloszony agentowi, który moze go obsluzyc np. poprzez modyfikacje swojej struktury meta-danych – adaptowalnosc do zmian, a wiec równiez odpornosc na sytuacje konfliktowe, jest elementem tezy pracy.

Proponowany model zaklada zatem funkcjonalny podzial procesu zbierania danych na dwie skladowe:

- konfigurowanie meta-danych – operacje zwiazane z rejestracja nowego zasobu w systemie, uzgodnieniem (stworzeniem lub rozszerzeniem) struktury atrybutów udostepnionych przez zasób do monitorowania,
- przekazywanie wartosci atrybutów – wprowadzanie wartosci konkretnych monitorowanych atrybutów.

Adaptowalnosc systemu polega na tym, ze moga sie one wzajemnie przeplatac. Konfiguracja meta-danych moze byc wykonywana dla danego zasobu w trakcie dzialania systemu wielokrotnie, ilekroc dany zasób usunie badz udostepni do monitorowania nowy atrybut.

3.4.3 Warstwa przechowywania danych

Warstwa przechowywania danych obejmuje system zapisu danych oraz wykorzystywana przez niego baze danych. System musi zapewnic okreslona przez interfejsy dostepu funkcjonalnosc. Równoczesnie jego wewnetrzna architektura oraz sposób komunikacji z baza danych, musi uwzgledniac nastepujace elementy:

- Wielodostep – system musi obslugiwac równoczesny dostep wielu dzialajacych równoczesnie agentów oraz klientów modulu prezentacji. Implementacja odpowiednich interfejsów musi uwzgledniac ten fakt zarówno pod wzgledem wydajnosci jak i synchronizacji dostepu.

- Integralnosc danych – system musi zapewniac integralnosc danych, w szczególnosci umozliwiac modyfikacje meta-danych, w trakcie pracy agentów wykorzystujacych te dane.
- Weryfikacja poprawnosci operacji zapisu – system musi sprawdzac, czy konstrukcja drzewa atrybutów i ich parametry definiowane poprzez interfejs zapisu sa poprawne i logicznie spójne z juz posiadanymi danymi.
- Kojarzenie danych – dane wprowadzane przez interfejs zapisu sa identyfikowane poprzez identyfikator atrybutu; system powinien dokonywac weryfikacji poprawnosci tego identyfikatora i odpowiednio kojarzyc wartosci z odpowiednim atrybutem.
- Wydajnosc i skalowalnosc – proces zapisu samych wartosci powinien byc mozliwie efektywny zarówno pod wzgledem czasu trwania zdalnego wywolania jak i czasu oraz zasobów potrzebnych na zapisanie danych.

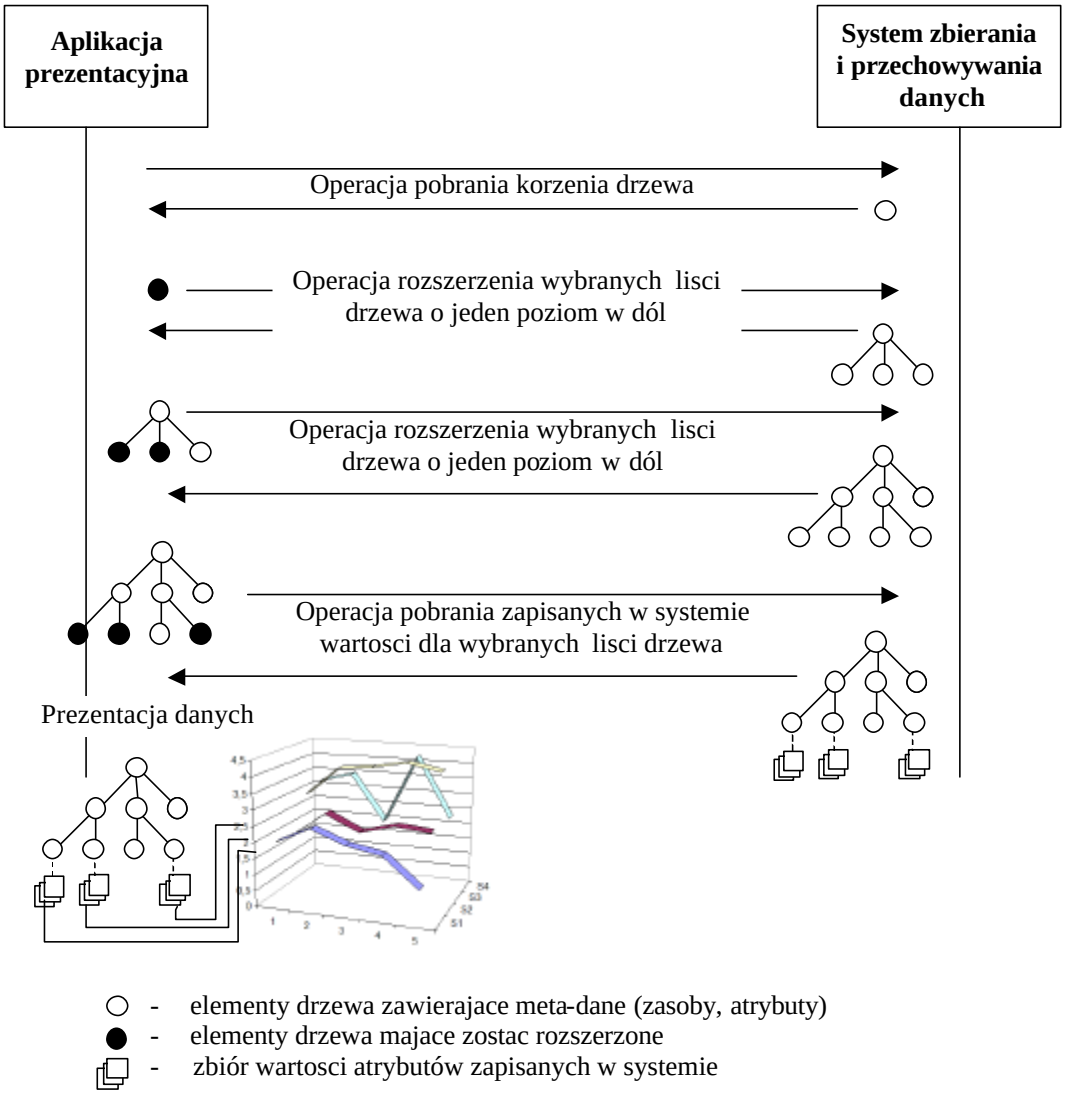
Realizacja powyzszych postulatów jest czesciowo wpisana w konstrukcje modelu, jest jednak silnie uwarunkowana przez przyjete dla realizacji systemu rozwiazania architektoniczne i projektowe. Zostaly one uwzglednione przy implementacji systemu, a kwestie skalowalnosci i wydajnosci beda weryfikowane doswiadczalnie w rozdziale 5.

3.4.4 Warstwa dostepu do danych

Głównym zadaniem warstwy dostepu do danych jest zapewnienie dostepu do meta-danych i danych w celu ich przegladania. Dostep ten musi uwzgledniac ich wzajemne relacje oraz dawac mozliwosc wyszukiwania i filtracji. Warstwa dostepu obejmuje interfejs dostepu oraz implementujace go elementy systemu.

Kluczowym elementem dla konstrukcji warstwy dostepu jest okreslenie mechanizmów pobierania danych z systemu, a konkretniej – sposobu formulowania zapytan oraz formatu odpowiedzi. Klasyczne systemy gromadzace dane pochodzace z systemów monitorowania przechowuja je w bazie danych, zgodnej ze standardem SQL; w jezyku SQL formulowane sa równiez zapytania. Dla tworzonego systemu zbierania i przechowywania danych takie rozwiazanie nie jest akceptowalne – wymagaloby od uzytkownika znajomosci wewnetrznej struktury danych, zaburzalo niezaleznosc warstw aplikacji i podwazalo sens stosowania obiektowego modelu danych.

Odpowiednikiem jezyka SQL dla obiektowych modeli danych sa obiektowe jezyki zapytan np. EJB-QL w J2EE, HQL w Hibernate [BK04]. Jezyki obiektowe nie sa tak elastyczne jak SQL. Ulatwiaja wyszukiwanie wlasciwych obiektów, jednak niezbyt dobrze radza sobie z operacjami agregacji danych – operacje te sa najczesciej wykonywane na obiektach dopiero po ich pobraniu z bazy. Najbardziej ogólny przypadek konstrukcji warstwy dostepu móglby bazowac na obiektowym jezyku zapytan, stworzonym w oparciu o zaproponowany obiektowy model danych. Konstrukcja i implementacja takiego jezyka jest jednak przedsiewzieciem bardzo skomplikowanym i sama w sobie pozostaje ciekawym tematem na rozprawe doktorska. Poniewaz glównym zadaniem warstwy dostepu jest umozliwienie dostepu do danych, mozna stworzyc rozwiazanie alternatywne, przedstawione ponizej.



Rys. 21 Dostep do danych – koncepcja uszczególawiania drzewa danych.

Pobieranie danych z systemu mozna calkowicie oprzec na ogólnym, obiektowym modelu danych. Rozwiazanie takie bazuje na koncepcji przekazywania tworzacej drzewo struktury

meta-danych miedzy aplikacja prezentacyjna, a systemem. Drzewo jest w kolejnych krokach uszczególawiane o jeden poziom w dól, rozwijane sa jednak jedynie wybrane liscie drzewa – Rys. 21. W ostatnim kroku, kiedy w lisciach drzewa sa juz atrybuty proste, wywolywana jest operacja pobrania danych – zwracane drzewo jest rozbudowywane o wartosci atrybutów i ma postac taka jak przedstawiona na Rys. 17 w punkcie 3.3.3. Operacja pobierania danych powinna miec dodatkowo parametry zwiazane z filtrowaniem wartosci ze wzgledu na znacznik czasowy oraz ilosc zwracanych wartosci.

Proponowane rozwiazanie jest wygodnym sposobem pobierania meta-danych i wartosci atrybutów, stosunkowo proste jest równiez stworzenie aplikacji prezentacyjnej dla tak przygotowanego drzewa. Nie rozwiazuje ono jednak kwestii agregacji danych po stronie systemu. Wartosci musza byc pobrane z systemu, przeslane do aplikacji prezentacyjnej i tam agregowane. Problem ten wystepuje jednak równiez w przytaczanych obiektowych jezykach zapytan, które wspieraja jedynie podstawowe operacje: MIN, MAX, SUM, AVG. Operacje te moga równiez stosunkowo latwo rozszerzyc proponowana koncepcje dzieki dodatkowemu sparametryzowaniu metody pobierania danych. Bardziej zlozone operacje agregacji beda musialy byc nadal realizowane przez aplikacje prezentacyjna, co wymaga przekazywania do niej duzej ilosci danych. Rozwiazaniem tego problemu jest komponentowa budowa systemu. Umozliwia ona tworzenie wyspecjalizowanych komponentów realizujacych zadana funkcjonalnosc i instalowania ich w systemie w trakcie jego dzialania. Nie jest przy tym wymagana zadna modyfikacja istniejacych wczesniej komponentów (interfejsów dostepu), bowiem mozliwe jest wspólistnienie kilku róznych interfejsów dostepu do danych w ramach jednego systemu. Wykorzystanie model dostepu opartego na drzewiastej strukturze danych jest podejsciem duzo prostszym, i to juz na poziomie koncepcyjnym, od tworzenia obiektowego jezyka zapytan. Daje przy tym porównywalne mozliwosci, a komponentowa konstrukcja systemu nie wyklucza stworzenia takiego jezyka w przyszlosci. Mozliwe jest ponadto instalowanie w czasie pracy systemu wyspecjalizowanych komponentów agregujacych o funkcjonalnosci niedostepnej dla wiekszosc obiektowych jezyków zapytan.

Ostatnim elementem warstwy dostepu jest kwestia protokolu dostepu. Termin protokól powinien byc w tym kontekscie rozumiany szerzej i obejmowac kilka mozliwych metod dostepu:

- HTTP/HTML – wymaga odpowiedniego interfejsu graficznego oraz niezbednych elementów po stronie serwera WWW,

- RMI – wymaga zestawu klas bedacego interfejsem programistycznym dostepu do systemu,
- HTTP/SOAP – umozliwiajacy integracje z innymi systemami, bazujacych np. na technologii WebService.

Przedstawiona koncepcja dostepu do danych jest dla wszystkich metod taka sama. Róznice, wynikajace ze specyfiki konkretnych rozwiazan, nie wplywaja na specyfikacje oraz implementacje interfejsów - moga one zostac prawie automatycznie wygenerowane na podstawie interfejsu programistycznego. Dlatego budowa warstwy dostepu zostala w kolejnych punktach przedstawiona w oparciu o interfejs programistyczny.

3.4.5 Wnioski

W punkcie 3.4 przedstawiono miejsce SZiPD w warstwowym modelu monitoringu. Omówiono oczekiwana funkcjonalnosc realizowana przez poszczególne warstwy oraz mechanizmy wymiany danych pomiedzy warstwami. Przedstawiono dwie koncepcje zapisu danych polegajace na przekazywaniu wartosci monitorowanych atrybutów lacznie oraz rozlacznie z opisujacymi je meta-danymi. Wybór koncepcji rozlacznego zapisu zostal poparty argumentami. Przedstawiono równiez wygodny sposób pobierania danych z systemu, oparty na drzewiastej strukturze meta-danych.

Omówione mechanizmy sa podstawa dla zdefiniowania w kolejnym punkcie uniwersalnych interfejsów dostepu do SZiPD, których stworzenie zaklada rozwiniecie tezy pracy.

3.5 Uniwersalne interfejsy dostepu

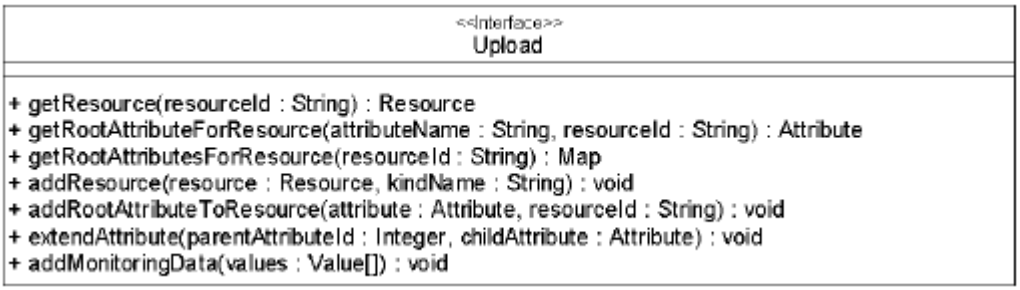
Opracowanie ogólnych interfejsów programistycznych, pozwalajacych na jednolity zapis oraz odczyt danych, pochodzacych z dowolnego z monitorowanych zasobów, jest waznym elementem rozwiniecia tezy pracy. W SZiPD wyrózniono dwa podstawowe interfejsy dostepu:

- interfejs zapisu danych i meta-danych,
- interfejs dostepu do danych i meta-danych.

Zostaly one zdefiniowane w tym punkcie.

3.5.1 Uniwersalny interfejs zapisu danych

Zadaniem uniwersalnego interfejsu zapisu danych jest umozliwienie zapisu danych zgodnych z ogólnym obiektowym modelem danych SZiPD. Tworzy go zbiór metod, jakie moga byc wywolane na systemie w celu zapisania danych – Rys. 22. Interfejs ten zapewnia mozliwosc zapisu konkretnych wartosci oraz konfiguracje meta-danych (danych opisujacych zasób i atrybuty), zgodnie z przyjetym w punkcie 3.4.2 rozlacznym trybem zbierania danych oraz ogólnym obiektowym modelem danych przedstawionym w punkcie 3.3.3.

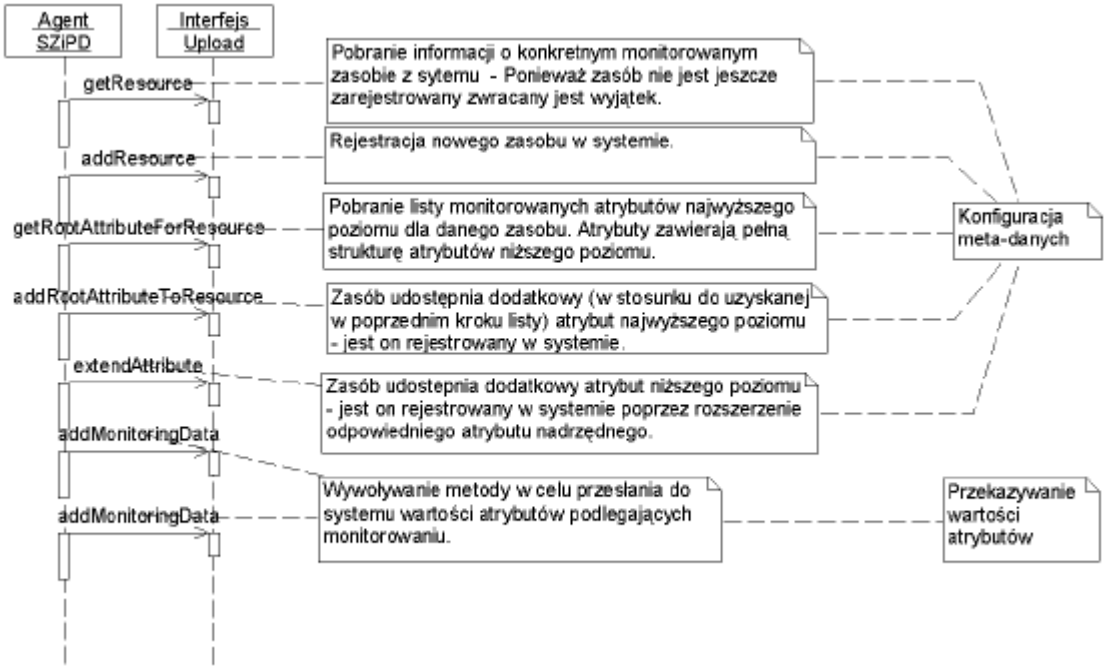


Rys. 22 Uniwersalny interfejs zapisu danych.

Metody interfejsu pozwalaja na podejmowanie nastepujacych dzialan:

- getResource() – metoda zwraca przechowywane w systemie informacje o konkretnym zasobie,
- getRootAttributesForResource() – metoda zwraca liste atrybutów najwyzszego poziomu dla podanego zasobu. Atrybuty te zawieraja w sobie wszystkie atrybuty nizszego poziomu. Jej dwie odmiany sluza do zwracania danych dla konkretnego atrybutu badz calej ich listy dla konkretnego zasobu,
- addResource() – metoda rejestruje w systemie nowy zasób,
- addRootAttributeToResource() – metoda rozszerza liste atrybutów najwyzszego poziomu dla danego zasobu,
- extendAttribute() – metoda rozszerza atrybut strukturalny na dowolnym poziomie zagniezdzenia o kolejny atrybut skladowy,
- addMonitoringData() – metoda powoduje dodanie do bazy zbioru danych bedacych argumentem jej wywolania.

Przykladowa sekwencje operacji wykonywanych na interfejsie zapisu danych obejmujaca konfiguracje meta-danych i przekazywanie wartosci obrazuje Rys. 23.

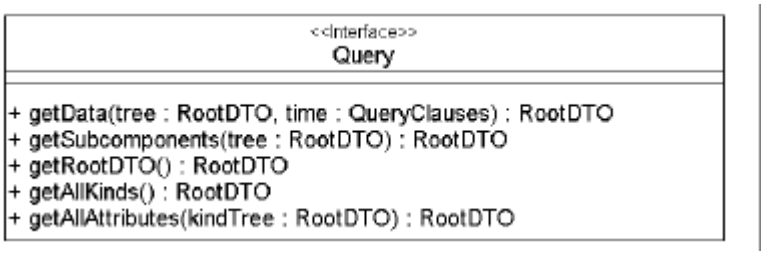


Rys. 23 Diagram sekwencji zapisu danych.

Proponowany interfejs zapisu danych posiada postulowane w tezie wlasnosci: monitorowany zasób moze w dowolnym momencie zmieniac (dodawac, usuwac) parametry jakie udostepnia do monitorowania, a pochodzace z zasobów dane moga byc zapisywane z rózna czestotliwoscia.

3.5.2 Uniwersalny interfejs do odczytu danych

Zadaniem interfejsu do odczytu danych jest zapewnienie elastycznego dostepu do zgromadzonych w systemie meta-danych i wartosci z uwzglednieniem ich wzajemnych relacji, umozliwienie selekcji oraz filtracji wartosci. Koncepcja pobierania danych z systemu, bazujaca na rozbudowie drzewa meta-danych zostala omówiona w punkcie 3.4.4. Interfejs potrzebny do realizacji tej koncepcji jest przedstawiony na Rys. 24.

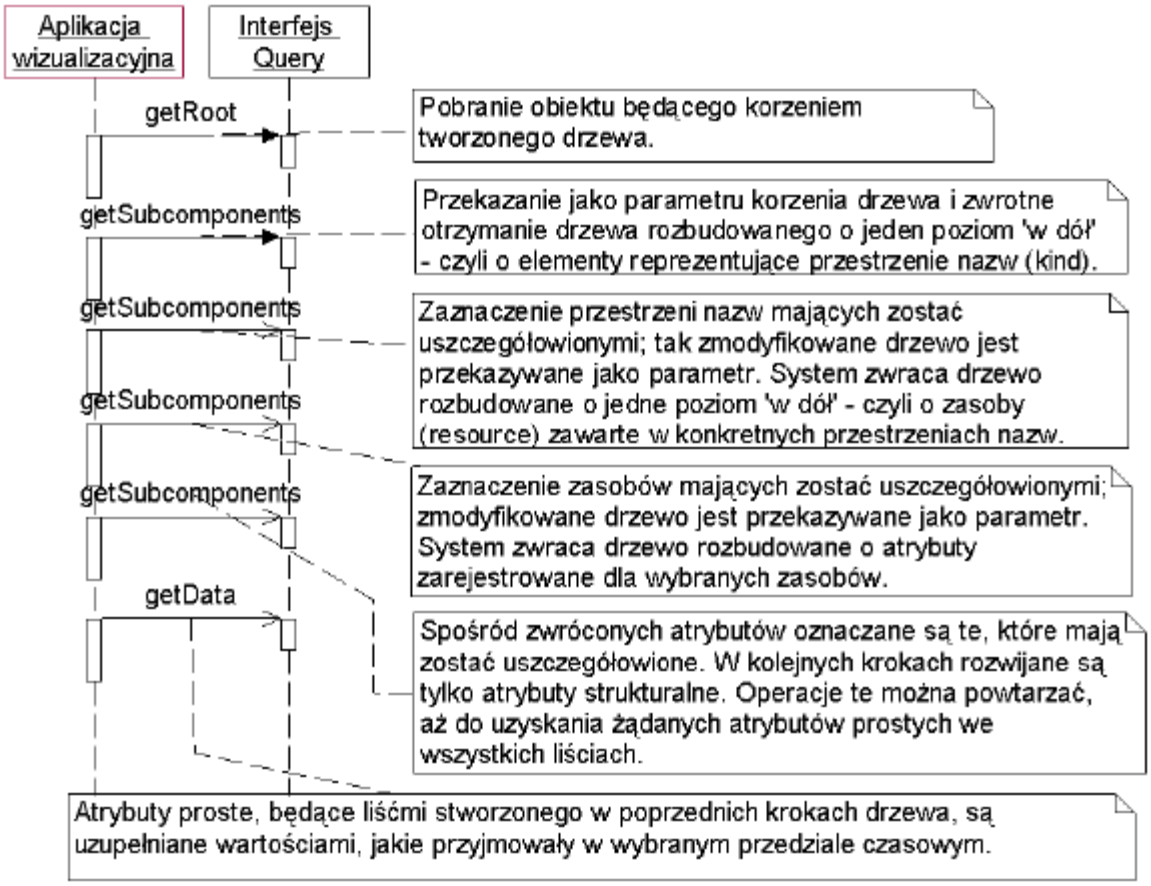


Rys. 24 Uniwersalny interfejs do odczytu danych.

Funkcjonalnosc poszczególnych metod interfejsu jest nastepujaca:

- getRootDTO() – zwraca korzen drzewa zapytan,
- getSubcomponent() – rozbudowuje drzewo o jeden poziom w dól,
- getData() – zwraca konkretne wartosci monitorowanych atrybutów dla wszystkich atrybutów prostych bedacych liscmi przekazanego jako parametr drzew, zgodnie z warunkami zawartymi w obiekcie klasy *QueryClauses* bedacego parametrem wywolania,
- getAllKinds() – zwraca drzewo zawierajace wszystkie przestrzenie nazw,
- getAllAttributes() – zwraca drzewo zawierajace wszystkie atrybuty nalezace bezposrednio do konkretnych przestrzeni nazw.

Metody interfejsu dzialaja w oparciu o implementacje ogólnego modelu danych, oparta na klasach kapsulkujacych – DTO (ang. data transfer object). Klasy te rozszerzaja ogólny model danych o mozliwosc zaznaczania elementów drzewa meta-danych do rozszerzenia oraz obiekt *RootDTO*, bedacy korzeniem drzewa.

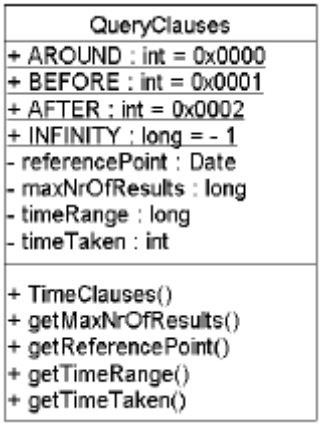


Rys. 25 Diagram sekwencji pobierania danych.

Przyklad wykorzystania uniwersalnego interfejsu dostepu do danych zostal przedstawiony na diagramie sekwencji pobierania danych – Rys. 25. Argumentem wywolywanej w kolejnych krokach operacji *getSubcomponents()* jest coraz wyzsze drzewo meta-danych z oznaczonymi do rozbudowy liscmi – co jest zgodne z koncepcja przedstawiona w punkcie 3.4.4. Ostatnia operacja, wywolywana na systemie w celu pobrania danych, jest *getData()*. Operacja ta przyjmuje jako parametr obiekt filtra, który umozliwia ograniczenie liczby zwracanych danych.

Filtracja danych

Podstawowa funkcjonalnosc w zakresie filtracji danych mozna uzyskac poprzez zastosowanie stworzonej w tym celu klasy *QueryClauses* – Rys. 26. Jest ona parametrem wywolania operacji *getData()* i umozliwia ograniczenie zwracanych wartosci ze wzgledu na czas ich utworzenia oraz ilosc.



Rys. 26 Klasa warunkujaca filtracje danych.

Mozliwe scenariusze wykorzystania tej klasy zakladaja podanie przedzialu czasu, z którego dane maja byc pobrane albo konkretnej chwili czasu wraz z wartoscia maksymalnego akceptowanego odchylenia od tej chwili. Ograniczenie ilosci zwracanych danych dotyczy ilosci wartosci zwracanych dla kazdego z atrybutów niezaleznie.

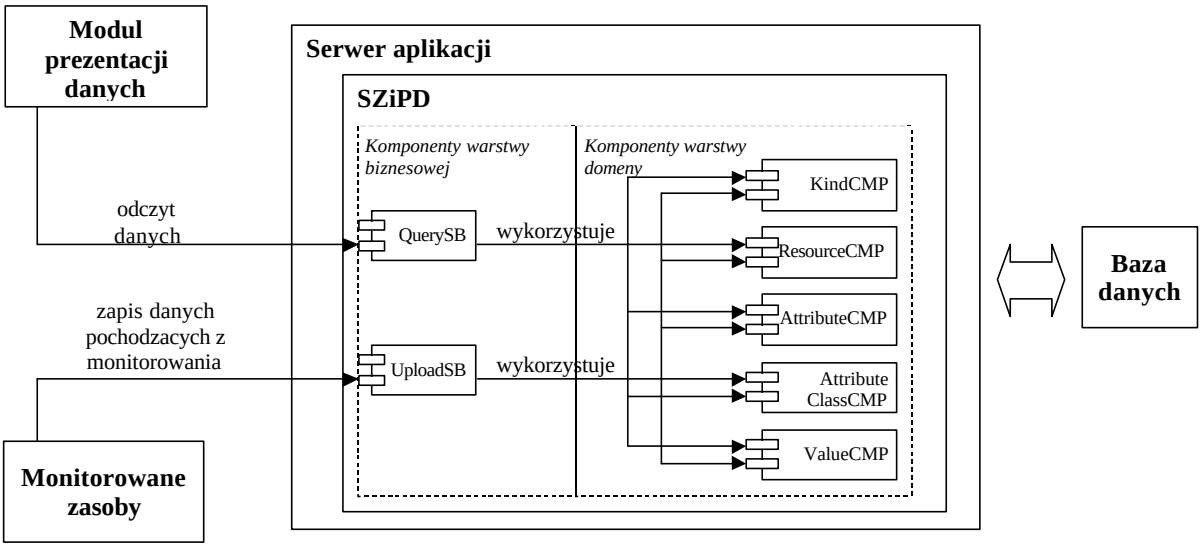
3.5.3 Wnioski

W punkcie 3.5 zostaly zdefiniowane uniwersalne interfejsy odczytu i zapisu danych oraz przedstawiono przykladowe diagramy sekwencji ich wykorzystania. Definicja tych interfejsów jest zwienczeniem rozwazan dotyczacych ogólnej architektury systemu, definicji ogólnego obiektowego modelu danych, mozliwosci wspólpracy pomiedzy SZiPD i agentami monitorujacymi oraz SZiPD a aplikacja monitorujaca. W celu wykazania poprawnosci

konstrukcji interfejsów, a tym samym waznej czesci tezy niniejszej pracy, autor zbuduje bazowy model systemu oraz dokona praktycznej weryfikacji jego funkcjonalnosci.

3.6 Bazowy model SZiPD

W tym punkcie przedstawiono wybrane aspekty implementacji bazowego modelu SZiPD w technologii komponentowej. Celem budowy systemu jest wykazanie poprzez implementacje realizowalnosci tezy pracy, mówiacej o mozliwosci konstrukcji w technologii komponentowej systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych.



Rys. 27 Bazowy model SZiPD.

Wewnetrzna budowa bazowego modelu SZiPD jest przedstawiona na Rys. 27. Architektura systemu jest zgodna z przyjeta w punkcie 3.2; system wykorzystuje ogólny model danych oraz implementuje uniwersalne interfejsy dostepu, zdefiniowane w punkcie 3.5.

Bazowy model SZiPD sklada sie z siedmiu komponentów; sa one przedstawione w dalszej czesci tego punktu. Konstrukcja modelu pomija pewne aspekty, które dla wykazania prawdziwosci tezy pracy uznano za drugorzedne:

- elementy bezpieczenstwa systemu - w tym uwierzytelnianie i autoryzacje,
- implementacje agentów oraz modulów instrumentacji, czyli elementów odpowiedzialnych za pozyskiwanie danych z monitorowanych zasobów,
- implementacje warstwy prezentacji.

Model bazowy jest uruchamiany na pojedynczej instancji serwera aplikacji. Rys. 27 wyróznia w tworzonym systemie komponenty warstwy biznesowej oraz warstwy domeny, podzial ten jest wykorzystany w kolejnych punktach w celu omówienia poszczególnych komponentów.

3.6.1 Komponenty warstwy biznesowej

W bazowym modelu SZiPD wyrózniono dwa komponenty warstwy biznesowej: UploadSB oraz QuerySB. Komponenty te implementuja uniwersalne interfejsy dostepu do systemu, zdefiniowane w punkcie 3.5; odpowiednio interfejs *Upload* oraz *Query.* Oba komponenty zostaly zaimplementowane jako sesyjne komponenty bezstanowe. Wybór ten wynikal z przeprowadzonej w punkcie 2.3.1 analizy mozliwosci komponentów EJB, majac na celu uzyskanie wysokiej wydajnosci oraz skalowalnosci – Tabela 1 w punkcie 2.3.2.

W celu zapewnienia funkcjonalnosci okreslonej w definicji uniwersalnych interfejsów dostepu komponenty warstwy biznesowej wykorzystuja komponenty warstwy domeny.

3.6.2 Komponenty warstwy domeny

W warstwie domeny bazowego modelu SZiPD wykorzystano piec komponentów. Glównym zadaniem tych komponentów jest zapis i odczyt z bazy danych stanu obiektów zgodnych z ogólnym modelem danych. Sposród, przedstawionych w punkcie 2.3.2, dostepnych komponentów warstwy domeny autor wybral komponenty encyjne zarzadzane przez kontener (CMP). Sa one, pomimo pewnych wad, najbardziej dojrzale wsród rozwiazan aktualnie wykorzystywanych w ramach technologii J2EE, a ich wlasnosci zwiazanych z mozliwoscia wspólpracy z róznymi bazami danych nie sposób nie docenic w kontekscie planowanych eksperymentów.

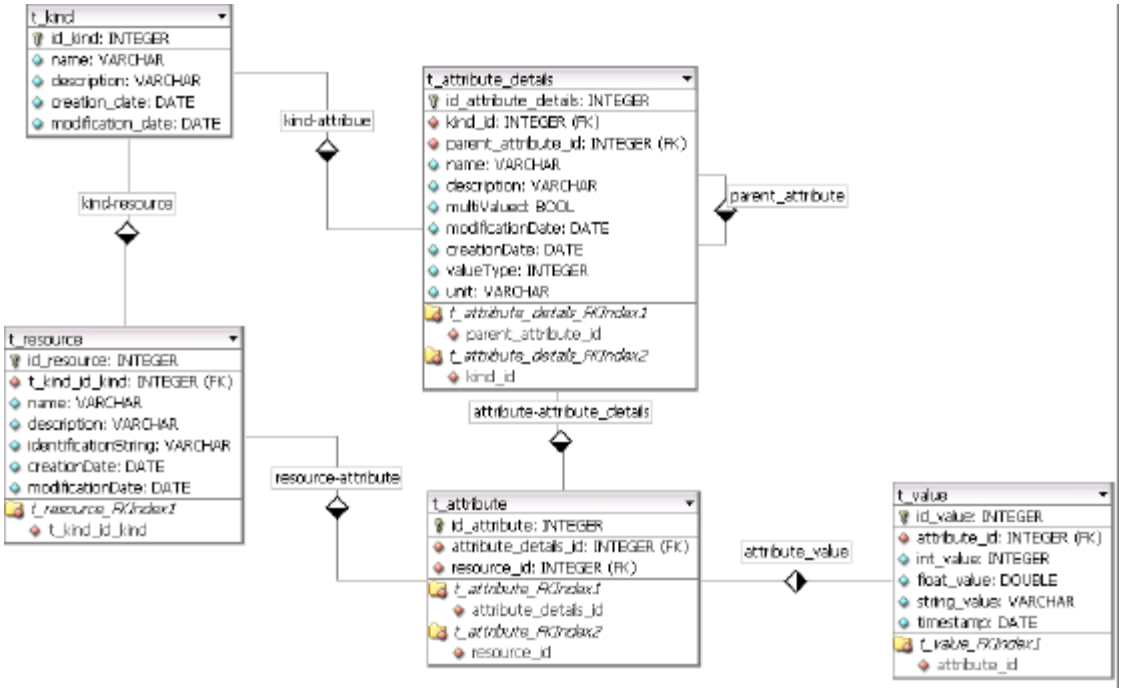
W systemie wyrózniono nastepujace komponenty:

- KindCMP – reprezentujacy klase *Kind,* zapisywany w tabeli *t_kind*,
- ResourceCMP – reprezentujacy klase *Resource,* zapisywany w tabeli *t_resource*,
- AttributeClassCMP –reprezentujacy klasy *SingleAttribute* i *StructureAttribute* wraz z polami dziedziczonymi z klasy *Attribute,* zapisywany w tabeli *t_attribute_details*,
- AttributeCMP – reprezentujacy klase *Attribute* i modelujacy powiazanie *ResourceCMP* i *AttributeClassCMP*, zapisywany w tabeli *t_attribute*,
- ValueCMP – reprezentujacy klase *Value,* zapisywany w tabeli *t_value*.

Wszystkie przedstawione powyzej komponenty posiadaja podstawowa funkcjonalnosc zwiazana z dostepem do swoich atrybutów oraz komponentów, z którymi pozostaja w relacji. Zdefiniowane w komponentach nazwy atrybutów jak i relacje sa analogiczne do zawartych w ogólnym modelu danych. Aby dane zawarte w komponentach mogly byc zapisane w relacyjnej bazie danych, dla przedstawionego modelu komponentowego opracowany zostal odpowiedni model relacyjny, który zostal przedstawiony w kolejnym punkcie.

3.6.3 Relacyjny model danych

Konstrukcja relacyjnego modelu danych jest niezbedna w celu przechowywania danych w relacyjnej bazie danych. Rys. 28 przedstawia relacyjny model danych wlasciwy dla ogólnego modelu danych oraz wyróznionych w systemie komponentów warstwy domeny.



Rys. 28 Relacyjny model danych.

Zaproponowany model relacyjny jest jednym z kilku, za pomoca których mozna poprawnie modelowac ogólny model danych pochodzacych z monitorowania systemów rozproszonych. Sygnalizowane w punkcie 2.3 problemy zwiazane z brakiem odpowiedników niektórych konstrukcji obiektowych w modelu EJB, rzutuja na zaproponowany relacyjny model danych systemu. Brak mechanizmów dziedziczenia i abstrakcji wymusil wprowadzenie tabel *t_attribute_details* oraz *t_attribute*, które zawieraja dane trzech klas: *Attribute*, *SingleAttribute*, *StructureAttribute*.

3.6.4 Weryfikacja kompletnosci rozwiazania

Wymagania funkcjonalne stworzonego bazowego modelu SZiPD zostaly zweryfikowane w kontekscie ich zgodnosci z wymaganiami dla systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych postawionymi w tezie pracy oraz uszczególowionymi na poczatku rozdzialu 3. Tabela 8 zbiera te wymagania funkcjonalne, okresla stopien ich spelnienia oraz specyfikuje mechanizmy realizacji; znak '✓' oznacza spelnienie przez SZiPD wymagania.

Wymaganie	Realizacja	Sposób realizacji
Róznorodnosc monitorowanych zasobów	✓	System poprzez wykorzystanie architektury opartej na agentach monitorujacych SZiPD moze wspólpracowac z róznymi zasobami.
Obiektowa reprezentacja danych	✓	System dziala w oparciu o ogólny model danych pochodzacych z monitorowania systemów rozproszonych.
Adaptowalnosc do zmian	✓	Obiektowa reprezentacja danych, wyróznienie meta-danych opisujacych monitorowane atrybuty gwarantuje adaptowalnosc systemu do zmian zbioru monitorowanych atrybutów w trakcie dzialania.
Dynamiczne definiowanie zasobów	✓	Uniwersalny interfejs zapisu umozliwia definiowanie nowych zasobów w trakcie dzialania systemu.
Dynamiczne definiowanie atrybutów	✓	Uniwersalny interfejs zapisu umozliwia definiowanie atrybutów w trakcie dzialania systemu.
Obsluga atrybutów zlozonych	✓	Ogólny model danych uwzglednia atrybuty zlozone (strukturalne). Zarówno interfejs zapisu jak i dostepu obsluguja atrybuty strukturalne.
Obsluga atrybutów wielowartosciowych	✓	Ogólny model danych uwzglednia atrybuty wielowartosciowe. Zarówno interfejs zapisu jak i dostepu obsluguja atrybuty wielowartosciowe.
Obsluga róznych typów danych	✓	Zostala przeprowadzona dyskusja typów danych, które powinny byc wspierane. Ogólny model danych wspiera postulowane typy danych.
Rózna czestotliwosc zapisu danych	✓	Interfejs zapisu przyjmuje dane w paczkach o dowolnej wielkosci. Agent SZiPD moze optymalizowac rozmiar paczki poprzez grupowanie danych przed wyslaniem.
Jednolity interfejs zapisu danych	✓	System posiada jeden interfejs zapisu dla róznorodnych zasobów, atrybutów i danych.

Dzialanie w róznych trybach monitorowania	✓	Sposób obslugi róznych trybów monitorowania jest zapewniany przez agenta SZiPD.
Jednolity interfejs dostepu do danych	✓	System posiada jeden interfejs dostepu dla róznorodnych zasobów, atrybutów i danych.
Selekcja i filtrowanie wartosci	✓	Poprzez mechanizm budowy drzewa danych interfejs dostepu posiada mechanizmy selekcji interesujacych uzytkownika danych wspiera równiez ich filtrowanie w dome nie czasu.
Komponentowa budowa	✓	Model zostal zbudowany w architekturze komponentowej zgodnej z J2EE.
Odpowiednia wydajnosc i skalowalnosc	?	Kwestiom wydajnosci i skalowalnosc jest w calosci poswiecony rozdzial 4.

Tabela 8 Weryfikacja zgodnosci modelu z przyjetymi zalozeniami.

3.7 Podsumowanie

W rozdziale 3 przedstawione zostaly zagadnienia konstrukcji systemu zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych. Zostala okreslona lista wlasnosci funkcjonalnych jakimi tworzony system powinien sie charakteryzowac. Lista ta obejmuje w szczególnosci wymagania odnosnie adaptowalnosci systemu do zmian zarówno rodzaju danych pochodzacych z konkretnego zasobu jak i trybu ich zbierania oraz konstrukcji uniwersalnych interfejsów zapisu i odczytu danych. Kolejne punkty okreslaja ogólna architekture systemu, mozliwosci integracji z istniejacymi agentami systemów monitorujacych oraz miejsce systemu w warstwowym modelu monitoringu. Autor proponuje ogólny obiektowy model danych dla danych pochodzacych z monitorowania systemów rozproszonych oparty o koncepcje meta-danych; definiuje uniwersalne interfejsy zapisu i odczytu danych oraz proponuje implementacje bazowego modelu systemu w technologii komponentowej.

Zamieszczona w punkcie 3.6.4, Tabela 8 weryfikuje uzyskane przez bazowy model systemu wlasnosci. Stworzony model posiada wszystkie wymagane wlasnosci funkcjonalne. Ostatnia pozostajaca do wykazania czescia tezy pracy jest stwierdzenie dotyczace wydajnosci i skalowalnosci stworzonego systemu. Kwestia ta jest przedstawiona w rozdziale 4 oraz zweryfikowana w sposób doswiadczalny opisany w rozdziale 5.

4 Rozszerzenia bazowego modelu SZiPD

„Sama wiedza nie wystarczy,
trzeba ja jeszcze umiec stosowac.”
- J. W. Goethe

Celem pracy jest wykazanie, ze w technologii komponentowej mozliwe jest stworzenie systemu zbierania i przechowywania danych zapewniajacego odpowiednia wydajnosc i skalowalnosc. Bazowy model systemu posiadajacy odpowiednie wlasnosci funkcjonalne zostal przedstawiony w punkcie 3.6. Niniejszy rozdzial poswiecony jest ocenie niefunkcjonalnych wlasnosci systemu: odpowiedniej wydajnosci oraz skalowalnosci. Termin wydajnosc zostal dla potrzeb tej pracy zdefiniowany w punkcie 2.4 jako maksymalna ilosc przetworzonych w jednostce czasu wywolan, przy których system zachowuje zadane parametry jakosciowe, obejmujace czas odpowiedzi oraz poprawnosc wykonania operacji. Skalowalnosc zostala zdefiniowana w rozdziale 1 jako zdolnosc systemu do utrzymania parametrów jakosciowych pomimo zwiekszania przetwarzanego strumienia danych, uzyskiwana poprzez rozszerzanie zasobów, w oparciu o które system dziala.

W celu uzyskania obu wlasnosci, przedstawiony w poprzednim rozdziale bazowy model SZiPD zostal poddany optymalizacji, a nastepnie szeregu modyfikacjom majacym na celu zrównoleglenie i uniezaleznienie przetwarzania we wszystkich jego warstwach. Kwestie jakie musialy zostac rozwiazane, obejmowaly:

- duza liczbe klientów, a wiec potrzebe równowazenia obciazenia poprzez transparentny rozdzial wywolan do róznych instancji serwerów aplikacji,
- bardzo duza ilosc danych, czyli zdolnosc systemu do przyjmowania milionów wartosci w ciagu godziny,
- optymalizacje procesu zapisu uwzgledniajaca rózny charakter danych i ich podzial na meta-dane i wartosci,
- uwzglednienie mozliwego, chwilowego wzrostu ilosci danych zapisywanych w systemie.

Autor zweryfikowal, czy i w jaki sposób przedstawione w punkcie 2.4 mechanizmy zwiekszania wydajnosci systemów komponentowych, poprawiaja parametry wydajnosciowe SZiPD. Przedstawione rozwiazania zostaly oparte o nastepujace koncepcje:

- optymalizacje procesu zapisu poprzez wykorzystanie specyficznych wlasnosci ogólnego modelu danych,
- zaawansowane mechanizmy klasteryzacji dostepne w serwerach aplikacji,
- mechanizmy komunikacji asynchronicznej oparte na brokerze komunikatów,
- mechanizmy partycjonowania danych i wykorzystanie kilku instancji baz danych.

Zaproponowane zostalo równiez rozwiazanie hybrydowe laczace kilka mechanizmów i potrafiace automatycznie adaptowac swoje dzialanie do wielkosci strumienia danych.

Przedstawione w kolejnych punktach modele zostaly przedstawione pod katem ich realizowalnosci jako rozszerzenie modelu bazowego, przedstawionego w punkcie 3.5. W celu wykonania implementacji i badan w sensownym czasie, autor pomija niektóre aspekty, stawiajac nastepujace zalozenia:

- do komunikacji wykorzystywane sa standardowe mechanizmy dostepne w jezyku Java (RMI, JDBC); nie podlegaja one optymalizacji,
- wykorzystywany jest wybrany serwer aplikacji, porównanie wydajnosci róznych serwerów oraz badanie mozliwosci ich wspólpracy lezy poza zakresem pracy,
- elementy infrastruktury systemu: system operacyjny, system plików, infrastruktura sieciowa posiadaja mechanizmy gwarantujace poprawna i wydajna prace w srodowisku klastra.

Wszystkie przedstawione w tym rozdziale modele zostaly przez autora zaimplementowane oraz zweryfikowane pod wzgledem funkcjonalnym – Tabela 8, zawarta w punkcie 3.6.4. Badaniom i testom porównawczym poswiecony jest rozdzial 5.

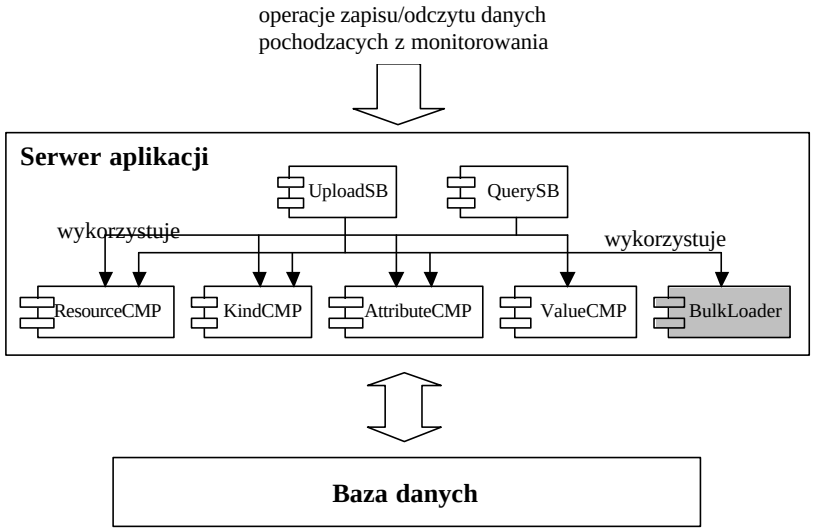
4.1 Optymalizacja modelu bazowego

Podstawowym dzialaniem podejmowanym w celu zwiekszenia wydajnosci systemu komputerowego jest próba optymalizacji jego dzialania w oparciu o posiadane zasoby sprzetowe. Optymalizacja bazowego modelu SZiPD polega w tym kontekscie na próbie przyspieszenia dzialania operacji wykonywanych przez komponenty, ze szczególnym uwzglednieniem operacji wykonywanych najczesciej.

W SZiPD najczesciej wykonywana operacja jest operacja zapisu paczki danych – *addMonitoringData()*, udostepniana przez komponent *UploadSB*, wykonywana wewnetrznie za posrednictwem komponentu *ValueCMP*. Specyfika dzialania komponentów warstwy domeny,

do których nalezy komponent *ValueCMP*, polega na indywidualnej obsludze kazdej zapisywanej wartosci, a tym samym kazdego wiersza bazy danych. Prowadzi to, bez wzgledu na przyjety sposób implementacji komponentu (CMP, BMP, OJB, DAO) do tworzenia duzej ilosci niezaleznych zadan zapisu danych do bazy danych, wykonywanych w ramach jednej operacji zapisu wywolanej przez klienta. Rozwiazanie oparte na komponentach CMP jak i inne, przedstawiane w czesci literaturowej pracy komponenty warstwy domeny, nie wspieraja operacji zapisu grupowego – Tabela 2 w punkcie 2.3.2.

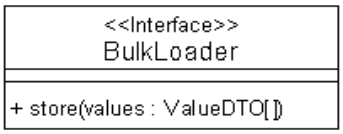
Proponowana modyfikacja modelu bazowego polega na wprowadzeniu komponentu dla grupowego zapisu danych o nazwie *BulkLoader*. Umozliwia on zapis w jednym wywolaniu operacji na bazie danych jednej badz wielu wartosci *ValueDTO* równoczesnie. Jest on wykorzystywany przez komponent *UploadSB* do efektywnego zapisu paczki danych otrzymanych od klienta. Sposób obslugi meta-danych oraz odczytu danych nie zmienia sie - Rys. 29.



Rys. 29 Zoptymalizowany model SZiPD.

4.1.1 Komponent dla grupowego zapisu danych

Wobec braku odpowiedniego wzorca projektowego J2EE dla sytuacji zapisu duzej ilosci danych, autor zdecydowal sie na odejscie od zasady jednolitej konstrukcji warstwy domeny poprzez wprowadzenie komponentu dla obslugi zapisu czesci danych. *BulkLoader* nie jest komponentem EJB, ale specjalizowana klasa umozliwiajaca zapis wartosci *ValueDTO* w bazie danych w ramach pojedynczej operacji na bazie danych.



Rys. 30 Interfejs komponentu BulkLoader.

Interfejs komponentu *BulkLoader* - Rys. 30, zawiera tylko jedna metode *store()*, wywolywana synchronicznie, w celu zapisania tablicy wartosci typu *ValueDTO*, bedacej argumentem wywolania tej metody. Wewnetrzna implementacja, wykorzystujaca bezposrednio mechanizm JDBC, tworzy na podstawie przekazanej tablicy wartosci zlozona instrukcje w jezyku SQL, która jest wykonywana na bazie danych w ramach jednego zdalnego wywolania.

Implementacja komponentu, pomimo bezposredniego wykorzystania wywolan SQL, jest przenosna pomiedzy róznymi silnikami baz danych. Zostala ona dodatkowo rozszerzona o obsluge specyficznych mechanizmów bazy danych Oracle, sluzacych do zapisu grupowego (ang. batch inserts), które sa automatycznie aktywowane w przypadku wykrycia współpracy z taka baza. Komponent moze zostac latwo rozszerzony o dalsze mechanizmy, specyficzne dla innych baz danych.

4.1.2 Generowanie kluczy głównych

Kolejna operacja, która jest bardzo czesto wykonywana w czasie dzialania SZiPD, jest generowanie kluczy głównych. Operacja ta jest wywolywana przy kazdej operacji zapisu danych i meta-danych.

Wykorzystanie w SZiPD komponentów typu CMP wymaga w fazie projektowej podjecia decyzji o sposobie generowania wartosci kluczy głównych, niezbednych do zapisania danych w bazie danych. Problem ten jest szeroko omawiany w wielu publikacjach [SW03] [RSB05] [Mar02] [KB02], bowiem specyfikacja EJB nie wspiera kluczy głównych generowanych po stronie bazy danych. Sposród mozliwych rozwiazan, autor wybral programowy generator sekwencji, z którego korzystaja wszystkie komponenty typu CMP. Podejscie takie gwarantuje przenosnosc systemu pomiedzy róznymi implementacjami serwerów aplikacji oraz bazami danych, nie jest jednak najbardziej wydajne.

Wprowadzenie komponentu *BulkLoader*, który nie jest oparty na komponentach CMP, umozliwia rezygnacje z generatora sekwencji dla zapisywanych wartosci i wykorzystanie mechanizmów generowania kluczy głównych po stronie bazy danych. Podejscie takie eliminuje problemy zwiazane z synchronizacja nadawania wartosci kluczom, zmniejsza ilosc operacji wywolywanych na bazie danych oraz nieznacznie obciazenie serwera aplikacji.

4.1.3 Wady i zalety modelu

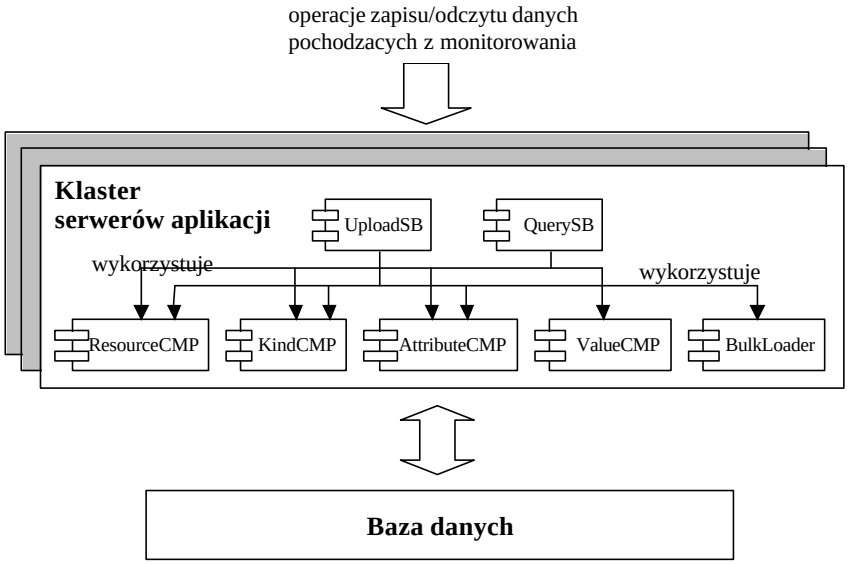
Stopien poprawy wydajnosci systemu, uzyskany poprzez wprowadzone optymalizacje zostal zweryfikowany doswiadczalnie, a uzyskane wyniki zbiera rozdzial 5. Wazna cecha architektury tak zoptymalizowanego systemu jest niejednolitosc budowy warstwy domeny oraz wykorzystanie mechanizmów automatycznego generowania kluczy glównych. Podkreslenia wymaga fakt, ze wprowadzenie komponentu dla zapisu grupowego jest realizowane przy zachowaniu deklaratywnego modelu transakcyjnosci wlasciwego dla J2EE oraz utrzymaniu wsparcia dla róznych silników baz danych.

Wada modelu jest uzaleznienie ilosci danych zapisywanych w jednej operacji *store()* komponentu *BulkLoader* od ilosci danych przesylanych przez klienta – otrzymany od klienta zbiór danych jest zapisywany w ramach jednej operacji na bazie danych. Bardziej rozbudowane implementacje komponentu moglyby dzielic ten zbiór na mniejsze, tak aby mial on optymalny rozmiar dla konkretnej bazy danych. Nie ma jednak prostego mechanizmu, który w ramach pojedynczego wywolania laczylby dane z róznych wywolan w celu ich dalszego zapisania w jednej operacji na bazie danych. Czesciowym rozwiazaniem dla tego problemu jest wlasciwa konfiguracja klientów – agentów SZiPD, którzy powinni przekazywac paczki danych o odpowiedniej objetosci, co daje dodatkowe korzysci w postaci zmniejszenia obciazenia medium, sygnalizowane w punkcie 3.2.

4.2 Model ze sklastrowanym serwerem aplikacji

Mozliwosci w zakresie skalowania aplikacji J2EE polegajace na klastrowaniu serwera aplikacji zostaly przedstawione w punkcie 2.4.2. Zwiekszenie wydajnosci w srodowisku klastra serwerów aplikacji jest realizowane poprzez odpowiednia konfiguracje serwerów oraz rozmieszczenie komponentów. Procesowi temu moga podlegac dowolne komponenty EJB, jednak uzyskiwany wzrost wydajnosci jest warunkowany rodzajem komponentu oraz sposobem jego implementacji.

Proces rozmieszczania komponentów w ramach klastra oraz ich konfiguracja jest specyficzna dla implementacji serwera aplikacji, generalnie jednak zaleca sie instalowanie wszystkich komponentów tworzacych aplikacje na kazdej instancji serwera aplikacyjnego tworzacego klaster. Podobne zalozenie przyjeto dla modelu SZiPD ze sklastrowanym serwerem aplikacji – Rys. 31.



Rys. 31 SZiPD dzialajacy na klastrze serwerów aplikacji.

Zlozony i zmudny proces instalowania wielu instancji serwerów aplikacji moze zostac usprawniony przez zastosowanie specjalizowanego oprogramowania zarzadzajacego klastrem serwerów. Wiekszosc serwerów aplikacji udostepnia zintegrowana konsole zarzadzajaca, poprzez która komponenty moga byc w sposób niemal dowolny rozmieszczone na wezlach klastra.

Dzialanie komponentów na klastrze serwerów aplikacji nie wymaga zmian w ich implementacji, a jedynie drobnych modyfikacji plików deskryptorów, co jest waznym atutem technologii J2EE. Wykorzystanie klastra serwerów aplikacji w istotny sposób wplywa na model wspólpracy poszczególnych instancji systemu ze wspóldzielonym zasobem, jakim jest baza danych oraz interakcje z klientami.

4.2.1 Wspólbiezny dostep do bazy danych

Dzialanie kilku instancji serwerów aplikacji powoduje wzrost ilosci równoczesnych polaczen z baza danych. Ilosc ta jest limitowana administracyjnie w bazie danych, w celu niedopuszczenia do nadmiernego obciazenia bazy danych,. Dostepne polaczenia stanowia zatem cenny zasób, którego przydzielanie powinno byc mozliwie automatycznie adaptowalne do aktualnego obciazenia i potrzeb wezlów. Dynamiczna, automatyczna konfiguracja ilosci polaczen do bazy danych posiadanych przez serwer aplikacji, jest realizowana poprzez mechanizm puli polaczen, przedstawiony w punkcie 2.4.3. Pule posiadaja dolne i górne ograniczenie swojej wielkosci oraz parametryzowane strategie zwiekszania i zmniejszania wielkosci, dzieki którym mozliwe jest dynamiczne zwiekszanie ilosci polaczen dostepnych w

puli oraz zwalnianie polaczen po ustalonym okresie ich niewykorzystywania. Strategia taka zapewnia bardziej optymalne wykorzystanie polaczen niz podejscie oparte na statycznym przypisaniu konkretnej ilosci polaczen do kazdego serwera aplikacji w klastrze – wlasciwe okreslenie parametrów puli polaczen pozostaje waznym elementem konfiguracji aplikacji w srodowisku klastra.

4.2.2 Równowazenie operacji wykonywanych przez klientów

Wykorzystanie klastra serwerów aplikacji powoduje istotne zmiany w sposobie dzialania aplikacji klienta, zwiazane z wyszukiwaniem komponentów oraz wywolywaniem operacji. Omówione w punkcie 2.4.2 rozwiazania z tego zakresu: HA-JNDI oraz HA-RMI bazuja na koncepcji obiektu posredniczacego swiadomego zwielokrotnienia (ang. replica aware stub). Gwarantuja one przezroczyste przelaczanie zadan klientów, zapewniaja równowazenie obciazenia i reakcje na sytuacje awaryjne na styku klient i serwer aplikacji – nie wymagaja przy tym zadnych zmian w kodzie aplikacji klienta.

Mechanizmy HA-JNDI oraz HA-RMI zostaly wykorzystane przy implementacji modelu SZiPD na klastrze serwerów aplikacji. Sposród dostepnych algorytmów przelaczania zadan klientów wybrany zostal algorytm „round robin”, co jest uwazane za wybór optymalny. Ewentualne bledne dzialanie tego mechanizmu skutkuje latwa do wykrycia sytuacja niedociazenia wezlów i równoczesnego przeciazenia innych, zmiana algorytmu jest mozliwa nawet w trakcie dzialania systemu.

4.2.3 Wady i zalety modelu

Klasteryzacja serwera aplikacji jest dzialaniem prowadzacym do zwiekszenia wydajnosci systemu. Jednak duza ilosc klientów wydajnie obslugiwanych poprzez sklastrowany serwer aplikacji, prowadzi do zwiekszenia ilosci zapisywanych w systemie danych, a tym samym do zwiekszania zapotrzebowania na ilosc polaczen oraz powaznego obciazenia bazy danych. W efekcie powstaje skalowalne rozwiazanie na poziomie komponentów, ale waskim gardlem pozostaje dostep do bazy danych.

Dostepny w serwerach aplikacji mechanizm puli polaczen do bazy danych nie zapewnia zwiekszenia wydajnosci bazy danych. W sytuacji braku wolnych polaczen w puli, operacja zapisu zwróci blad lub moze zostac wstrzymana do momentu zwolnienia polaczenia – prowadzi to do drastycznego wydluzenia czasu wykonania operacji, a tym samym pogorszenia parametrów jakosciowych pracy systemu. Powiekszanie puli polaczen bez

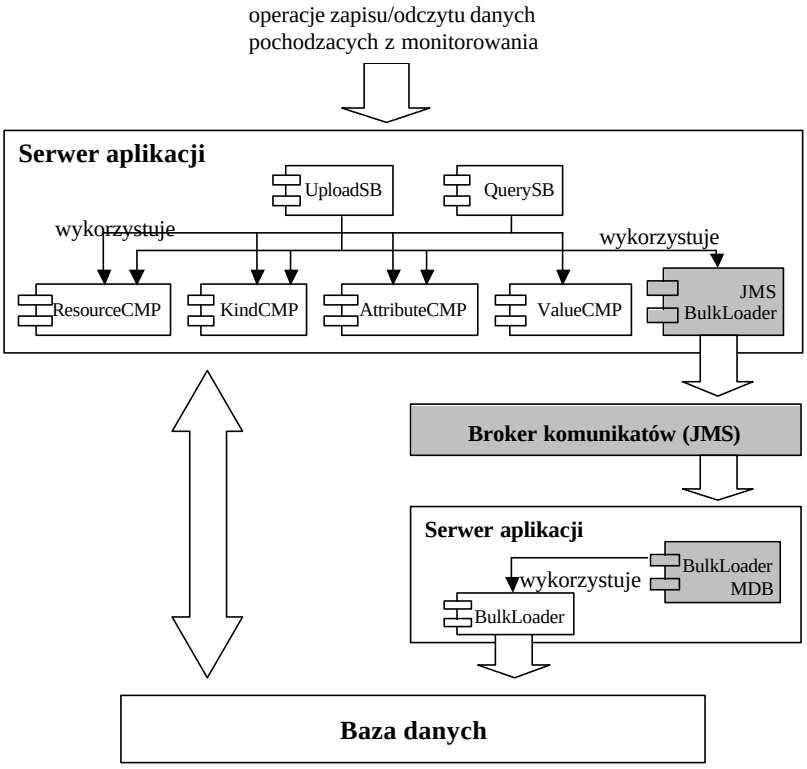
ograniczen nie jest mozliwe poniewaz skutkuje przeciazeniem serwera bazy danych, czesto równiez napotyka na ograniczenia licencyjne. Powyzsze uwarunkowania prowadza równiez do wniosku, ze o ile jest mozliwe stosunkowo latwe uruchomienie aplikacji J2EE na klastrze serwerów aplikacji, to uzyskanie wysokiej wydajnosci wymaga zlozonego strojenia licznych parametrów konfiguracyjnych serwera.

Kluczowym dla osiagniecia skalowalnosci systemu jest szukanie wydajnego, skalowalnego rozwiazania dla operacji zapisu danych w bazie danych, uwzgledniajacego mozliwe chwilowe wzrosty w ilosci zapisywanych danych (piki w obciazeniu) oraz synchroniczna nature wywolan operacji na komponentach sesyjnych.

4.3 Model oparty na brokerze komunikatów

Koncepcja zastosowania brokera komunikatów w celu poprawienia wydajnosci systemu komponentowego zostala przyblizona w punkcie 2.4.4. Opisane tam podstawowe wlasnosci zastosowania brokera: asynchronicznosc i kolejkowanie operacji oraz równowazenie obciazenia sa bardzo pozadane dla operacji o dlugim czasie wykonania. Z taka sytuacja mamy do czynienia w SZiPD. Operacja zapisu paczki danych do bazy danych w przypadku duzej ilosci danych jest dlugotrwala; jej wykonanie wymaga ponadto uzyskania dostepu do polaczenia z baza danych, na którego zwolnienie – w przypadku duzego obciazenia systemu – trzeba oczekiwac.

Koncepcja zastosowania sytemu kolejkowego w SZiPD rozszerza model systemu o element brokera komunikatów, komponent przekazujacy dane do kolejki brokera oraz komponent sterowany komunikatami przetwarzajacy komunikaty z kolejki, zostaly one przedstawione na Rys. 32. Model SZiPD wykorzystujacy broker komunikatów wymaga stworzenia komponentu *JMSBulkLoader*, którego zadaniem jest stworzenie komunikatu na podstawie zbioru obiektów *ValueDTO* i jego umieszczenie w dedykowanej kolejce brokera komunikatów dzialajacej w modelu punkt-punkt. Komunikat jest nastepnie przetwarzany przez komponent *BulkLoaderMDB*; pobierany jest z niego zbiór obiektów *ValueDTO* i zapisywany w bazie danych z wykorzystaniem, znanego z poprzedniego modelu, komponentu *BulkLoader*.



Rys. 32 Ogólny model SZiPD wykorzystujacy broker komunikatów.

Wprowadzenie brokera komunikatów umozliwia rozluznienie zaleznosci pomiedzy komponentami przyjmujacymi wywolania od klientów, a komponentami zapisujacymi dane w bazie, powoduje zwiekszenie zrównoleglenia przetwarzania oraz uniezaleznia zakonczenie wykonywania operacji przez klienta od obciazenia bazy danych.

4.3.1 Komponent sterowany komunikatami

Rola komponentu sterowanego komunikatami jest przetwarzanie komunikatów przyjmowanych z kolejki brokera komunikatów. Zgodnie z wymaganiami architektury J2EE dla tego typu komponentów (punkt 2.3.1) komponent taki musi implementowac metode *onMessage()*, która jest wywolywana ilekroc komponent ma przetworzyc komunikat. Interfejs komponentu *BulkLoaderMDB* zawiera wylaczenie te metode – Rys. 33.

Rys. 33 Interfejs komponentu BulkLoaderMDB.

Dzialanie metody polega na przyjeciu komunikatu, wyekstrahowaniu z niego zbioru obiektów *ValueDTO*, a nastepnie wywolaniu metody *store()* komponentu *BulkLoader*, którego dzialanie przedstawiono w punkcie 4.1.1. Zakonczenie wywolania metody *store()* oznacza poprawne

zapisanie danych w bazie, komunikat zostaje uznany za przetworzony. W razie wystapienia wyjatku w trakcie zapisu (np. w sytuacji niedostepnosci bazy danych), komponent nie potwierdza przetworzenia komunikatu, który pozostaje w kolejce brokera. Czas, po jakim broker ponownie zleci jego przetworzenie oraz maksymalna ilosc prób, sa parametrami konfiguracyjnymi kolejki komunikatów. W obu powyzszych przypadkach komponent powraca do stanu oczekiwania na nastepne wywolania.

4.3.2 Trwalosc operacji zapisu

Wykonywanie synchronicznej operacji *addMonitoringData()* daje gwarancje jej poprawnego wykonania lub uzyskania informacji o bledzie w przetwarzaniu. Wykorzystanie brokera komunikatów powoduje, ze operacja *addMonitoringData()* jest asynchroniczna, a jej poprawne wykonanie oznacza jedynie potwierdzenie przyjecia komunikatu do kolejki. Trwaloscia elementów w kolejce zarzadza element brokera, nazwany menadzerem zapisu (ang. persistent manager). W zaleznosci od konfiguracji, modul ten przechowuje dane w pamieci operacyjnej, pliku lub bazie danych; warunkuje to mozliwosc przetworzenia nieprzetworzonych komunikatów pomimo awarii systemu oraz bardzo istotnie wplywa na wydajnosc. Stosowanie menadzera zapisu opartego o pamiec operacyjna skutkuje, w razie wystapienia sprzetowej lub programowej awarii, utrata wszystkich nieprzetworzonych komunikatów znajdujacych sie w kolejce, jest jednak duzo bardziej wydajne.

Poprawa wydajnosci systemu nie moze byc realizowana kosztem narazenia uzytkownika na bledne dzialanie systemu lub utrate danych. Dlatego w SZiPD broker komunikatów zapisuje dane do podrecznego pliku, co gwarantuje, ze dane przyjete do zapisania nie zostana utracone w razie awarii systemu. W przypadku stosowania jednej instancji brokera komunikatów zapis ten pozostanie waskim gardlem systemu, podobnie jak w modelu poprzednim, czego autor chcial uniknac. Uzyskanie pozadanych wlasnosci w proponowanym modelu jest zatem mozliwe jedynie w oparciu o kilka instancji brokerów komunikatów.

4.3.3 Rozmieszczenie i klasteryzacja elementów systemu

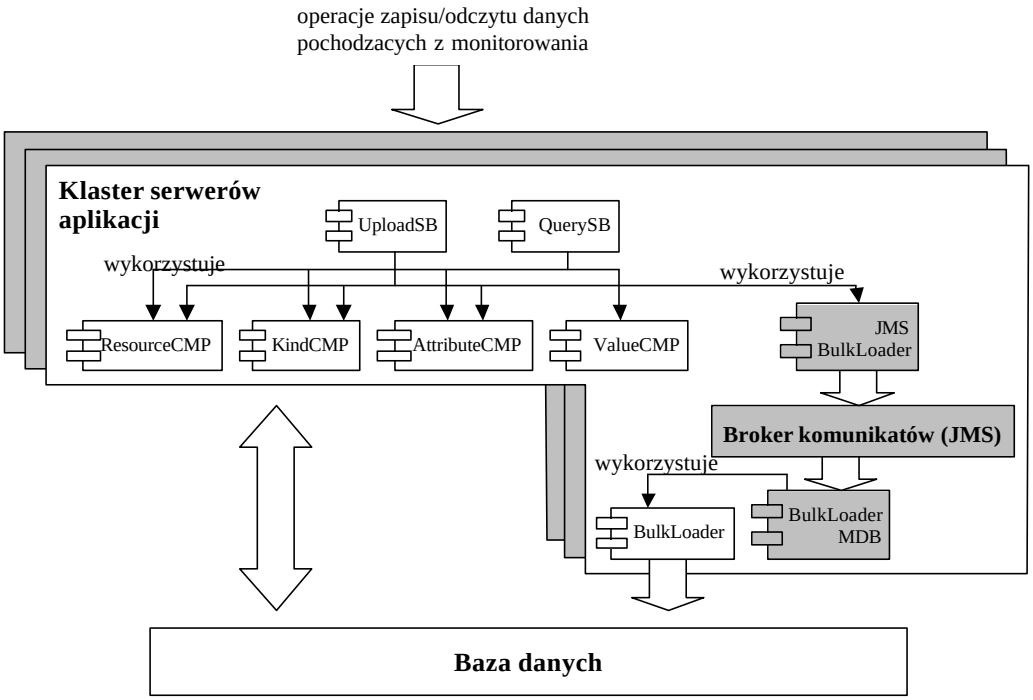
Komponenty *BulkLoaderMDB* moga dzialac w ramach tego samego serwera aplikacji co pozostale komponenty systemu. Ich ilosc moze byc optymalnie dobrana do ilosci dostepnych polaczen z baza danych przy uwzglednieniu jej optymalnego obciazenia. Maksymalizuje sie w ten sposób ilosc danych zapisywanych w bazie danych w jednostce czasu.

4.3.4 Zintegrowany broker komunikatów

W proponowanym modelu, wydajne przyjmowanie komunikatów wymaga wykorzystania kilku instancji brokera komunikatów. Aby ustalic ich ilosc w relacji do ilosci instancji aplikacji nalezy ocenic nastepujace, mozliwe przypadki:

- wykorzystanie kilku brokerów do obslugi jednej instancji aplikacji,
- wspóldzielenie kilku brokerów przez kilka instancji aplikacji,
- przydzielenie jednego brokera do jednej instancji aplikacji.

Sposród tych trzech mozliwosci, do celów implementacyjnych autor wybral ostatnia z nich. Jest ona pozornie najbardziej restrykcyjna, celem prac jednak jest znalezienie rozwiazania wydajnego, a nie najbardziej elastycznego. Przydzielenie jednego brokera do jednej instancji aplikacji, a tym samym jednej instancji serwera aplikacji, upraszcza budowe komponentu *JMSBulkLoader*, który przekazuje komunikaty do jednego brokera i nie musi implementowac mechanizmów ich rozdzielania. Przede wszystkim jednak, podejscie to umozliwia uzycie brokerów komunikatów zintegrowanych z serwerem aplikacji i wykorzystanie wydajnych odwolan lokalnych – Rys. 34.



Rys. 34 SZiPD wykorzystujacy zintegrowany broker komunikatów.

Proponowany model SZiPD wykorzystujacy zintegrowany broker komunikatów zapewnia optymalne obciazenie zasobów sprzetowych; nie wymaga przy tym wykorzystania zadnych

dodatkowych mechanizmów równowazenia obciazenia. Wszelkie sytuacje przeciazenia serwera aplikacji sa obslugiwane na styku klient – serwer aplikacji. Te same mechanizmy umozliwiaja dynamiczne zwiekszanie ilosci brokerów poprzez uruchamianie nowych, niezaleznych instancji serwerów aplikacji. Integracja zapewnia unikniecie sytuacji niedociazenia wezlów klastra serwera aplikacji przy przeciazeniu wezlów klastra brokera komunikatów i odwrotnie, co jest trudne do unikniecia w przypadku rozdzielenia systemów. W porównaniu z modelem ogólnym – Rys. 32, uproszczone zostaje zarzadzanie oraz zredukowana liczba zdalnych wywolan.

4.3.5 Wady i zalety modelu

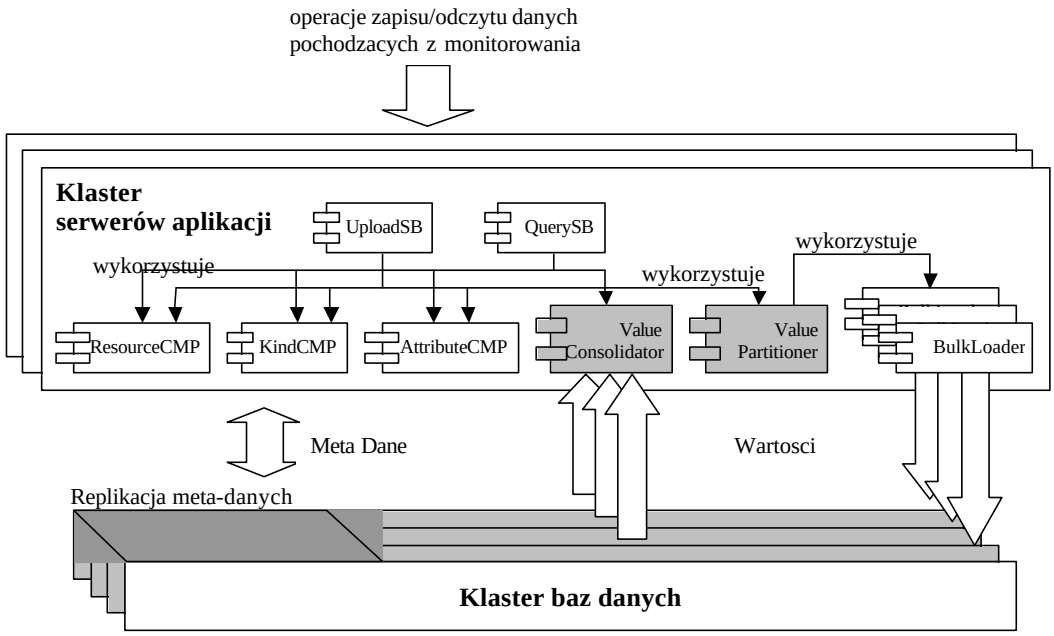
Do zalet modelu nalezy zaliczyc rozwiazanie problemu chwilowego wzrostu ilosci danych naplywajacych do systemu (piki w obciazeniu). Wprowadzenie mechanizmów kolejkowania zapewnia mozliwosc utrzymania czasu wykonania operacji zapisu na stalym poziomie. „Nadwyzka” danych jest przechowywana przez broker komunikatów i zapisywana w bazie danych gdy jest ona mniej obciazona. Prowadzi to do optymalnego wykorzystania bazy danych, a wiec pracy przy obciazeniu zapewniajacym zapis najwiekszej ilosci danych – nie podnosi jednak ilosci danych, jakie baza jest w stanie zapisac w jednostce czasu (ang. throughput). Wartosc ta pozostaje granicznym parametrem wydajnosci systemu. Naplywanie do systemu wiekszego strumienia danych przez dluzszy okres skutkuje odmowa przyjecia danych, zwiazana z przekroczeniem dopuszczalnych dlugosci kolejek w brokerach oraz bardzo duzymi opóznieniami w zapisie danych.

Wada modelu jest wprowadzenie opóznienia w zapisie danych, zakonczenie wykonania operacji zapisu przez klienta nie oznacza bowiem ich zapisania w bazie; dodatkowo opóznienie to jest zmienne i zalezy od obciazenia systemu. Mimo, iz broker komunikatów dziala w ramach serwera aplikacji, wykonanie modelu wymaga istotnych zmian w implementacji komponentów oraz zlozonej konfiguracji srodowiska uruchomieniowego.

4.4 Model oparty na partycjonowaniu danych

Jedynym sposobem pokonania granicy wydajnosci jednej instancji bazy danych jest wykorzystanie klastra baz danych. Sposród dostepnych rozwiazan, najwyzsza skalowalnosc zapewnia architektura bez współdzielen (ang. shared nothing), której wykorzystanie wymaga partycjonowania danych – punkt 2.4.3.

Proponowane wczesniej modele traktowaly baze danych jako system jednolity (SSI – ang. single system image). Konfiguracja bazy danych mogla zatem wewnetrznie wykorzystywac mechanizmy klasteryzacji, jednak dla zewnetrznych systemów pozostawala jednolitym, spójnym elementem infrastruktury. Takie „widzenie" bazy danych jest przyjete dla systemów klasy J2EE; umozliwia ono wykorzystanie mechanizmów partycjonowania, jednak ich zastosowanie jest warunkowane ich dostepnoscia w wykorzystywanej bazie danych (wiekszosc popularnych, dostepnych na licencji otwartej baz nie wspiera tego mechanizmu). Aby wykorzystac mechanizmy partycjonowania w sposób niezalezny od bazy danych, w punkcie 2.4.3 zaproponowano model 'swiadomy partycjonowania'. Model SZiPD bazujacy na klasterze baz danych oraz mechanizmach partycjonowania danych przedstawia Rys. 35.



Rys. 35 SZiPD wykorzystujacy partycjonowanie danych.

Proponowany model wykorzystuje klaster bazy danych, w którym meta-dane sa replikowane miedzy instancjami, a zawarte dotychczas w jednej tabeli wartosci rozdzielone i zapisywane w róznych instancjach. Podejscie takie wymaga stworzenia dwóch nowych komponentów *ValuePartitioner* i *ValueConsolidator* odpowiedzialnych odpowiednio za partycjonowanie danych wedlug przyjetego klucza oraz za pobieranie spelniajacych zadania odczytu danych z róznych instancji bazy danych.

4.4.1 Komponent partycjonujacy dane

Zadaniem komponentu *ValuePartitioner* jest zapis danych w kilku instancjach baz danych. Komponent ten posiada jedna publiczna metode *store()*, której argumentem jest zbiór

obiektów *ValueDTO*. Jej wywolanie powoduje zapis danych w róznych instancjach baz danych z wykorzystaniem wielu instancji komponentu *BulkLoader*, opisanego w punkcie 4.1.1.

Najistotniejszym zagadnieniem przy konstrukcji komponentu jest kwestia sposobu partycjonowania danych oraz wyboru instancji bazy danych, do której maja one zostac zapisane; mozliwe scenariusze obejmuja:

- róznicowanie na podstawie meta-danych – podzial danych ze wzgledu na kontekst interpretacyjny; dane opisujace konkretny zasób moga byc zapisywane w tej samej instancji bazy danych,
- róznicowanie na podstawie typu danych – przedstawiony w punkcie 3.3.2 model danych wyróznial trzy typy danych, moze on zostac wykorzystany jako klucz przy podziale danych miedzy instancje baz danych,
- wykorzystanie szeregowania cyklicznego – czyli cykliczny zapis do kolejnych instancji oparty na algorytmie „round robin”.

Powyzsze mozliwosci nalezy rozwazyc w kontekscie ich skalowalnosci, równomiernosci rozkladu danych, latwosci dodawania kolejnych instancji baz danych, zlozonosci implementacji algorytmu partycjonujacego oraz wplywu na komponent *ValueConsolidator*, odpowiedzialny za pózniejsze scalanie danych rozproszonych w róznych instancjach.

Zdaniem autora, najkorzystniejszym wyborem jest szeregowanie cykliczne, posiada ono najlepsze wlasnosci w zakresie skalowalnosci, równomiernosci rozkladu danych oraz latwosci dodawania kolejnych instancji baz danych. Jego minusem jest potrzeba przeszukiwania wszystkich instancji w celu znalezienia danych pasujacych do zapytania. Jednak, jesli uwzgledni sie mozliwosc rozszerzania zbioru instancji baz danych w czasie dzialania systemu, to problem ten wystepuje równiez w przypadku pozostalych rozwiazan.

Komponent *ValuePartitioner* zostal stworzony na bazie szeregowania cyklicznego, i dziala w oparciu o wspóldzielona, cykliczna liste komponentów *BulkLoader*. Parametr wywolania metody *store()*, bedacy zbiorem obiektów *ValueDTO,* jest dzielony na podzbiory o krotnosci równej optymalnej ilosci danych, zapisywanych w obslugiwanej bazie danych w jednej komendzie jezyka SQL. Kazdy z podzbiorów jest niezaleznie zapisywany przez kolejny, pobrany z listy, komponent *BulkLoader*.

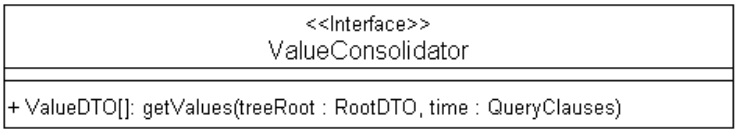
Poniewaz instancja komponentu *BulkLoader* zapisuje dane w jednej instancji bazy danych, stopien obciazenia konkretnej instancji bazy danych jest uzalezniony od ilosci komponentów

BulkLoader z nia polaczonych. Ilosc tych komponentów powinna byc dostosowana do ilosci dostepnych polaczen do bazy danych; sa one dzielone pomiedzy instancje serwerów aplikacji oraz udostepniane poprzez mechanizm pul polaczen. Przypisanie komponentu do konkretnej instancji bazy danych jest parametrem konfiguracyjnym komponentu i moze byc zmieniane w czasie dzialania systemu. Stosunkowo latwo jest zatem regulowac obciazenie poszczególnych instancji baz danych poprzez zmiane ilosci komponentów polaczonych z konkretna instancja. Jest to szczególnie przydatne w przypadku instancji baz danych o silnie zróznicowanej wydajnosci.

4.4.2 Komponent laczacy dane

Komponenty encyjne zarzadzane przez kontener (CMP) moga wykorzystywac jedno zródlo danych. W sytuacji rozproszenia wartosci pomiedzy kilka instancji baz danych, dostep poprzez komponent CMP nie jest mozliwy. Charakterystyczne dla podejscia CMP indywidualne odczytywanie wartosci musi zostac zastapione operacjami grupowymi.

Rola komponentu *ValueConsolidator* jest dostarczenie mechanizmów grupowego odczytu danych z wielu instancji baz danych, zgodnie z interfejsem przedstawionym na Rys. 36.



Rys. 36 Interfejs komponentu ValueConsolidator.

Metoda *getValues()* tworzy zapytanie SQL z przyjetej jako parametr struktury atrybutów i filtra danych,. Zapytanie to jest w takiej samej postaci wykonywane na wszystkich instancjach baz danych. Uzyskane wyniki sa konwertowane na obiekty *ValueDTO* i umieszczane we wspólnej tablicy, która jest zwracana jako wynik dzialania metody. Dzialanie takie wymaga wykorzystania puli polaczen z bazami danych, tak aby konkretne polaczenie bylo zajmowane wylacznie na czas wykonywania operacji przeszukania.

4.4.3 Replikacja meta-danych

Dobre praktyki tworzenia aplikacji, implementujacych mechanizmy odwzorowania modelu obiektowego na relacyjny, zalecaja weryfikacje danych zarówno w modelu obiektowym jak i relacyjnym. Baza danych powinna kontrolowac spójnosc danych, okreslona przez wiezy integralnosci zawarte w modelu relacyjnym. Kontrola taka jest stosunkowo

prosta w przypadku jednej instancji bazy danych, moze sie powaznie skomplikowac w przypadku klasteryzacji bazy i wprowadzeniu mechanizmów partycjonowania danych.

W modelu SZiPD wykorzystujacym partycjonowanie danych, kluczowym problemem jest utrzymanie integralnosci relacji pomiedzy danymi, a meta-danymi. Wobec przyjetego algorytmu partycjonowania, kontrola integralnosci taka wymaga posiadania kompletnej struktury meta-danych w kazdej instancji bazy danych, a wiec replikacji meta-danych miedzy instancjami. Replikacja moze zostac zrealizowana poprzez dostepne w bazach danych mechanizmy replikacji lub zewnetrzne mechanizmy replikujace, takie jak opisana w punkcie 2.4.3 biblioteka G-JDBC. Mechanizmy te dzialaja w oparciu o rozglaszanie grupowe, ich zlozonosc jest zatem niezalezna od ilosci instancji baz danych i nie powinna istotnie wplywac na skalowalnosc systemu.

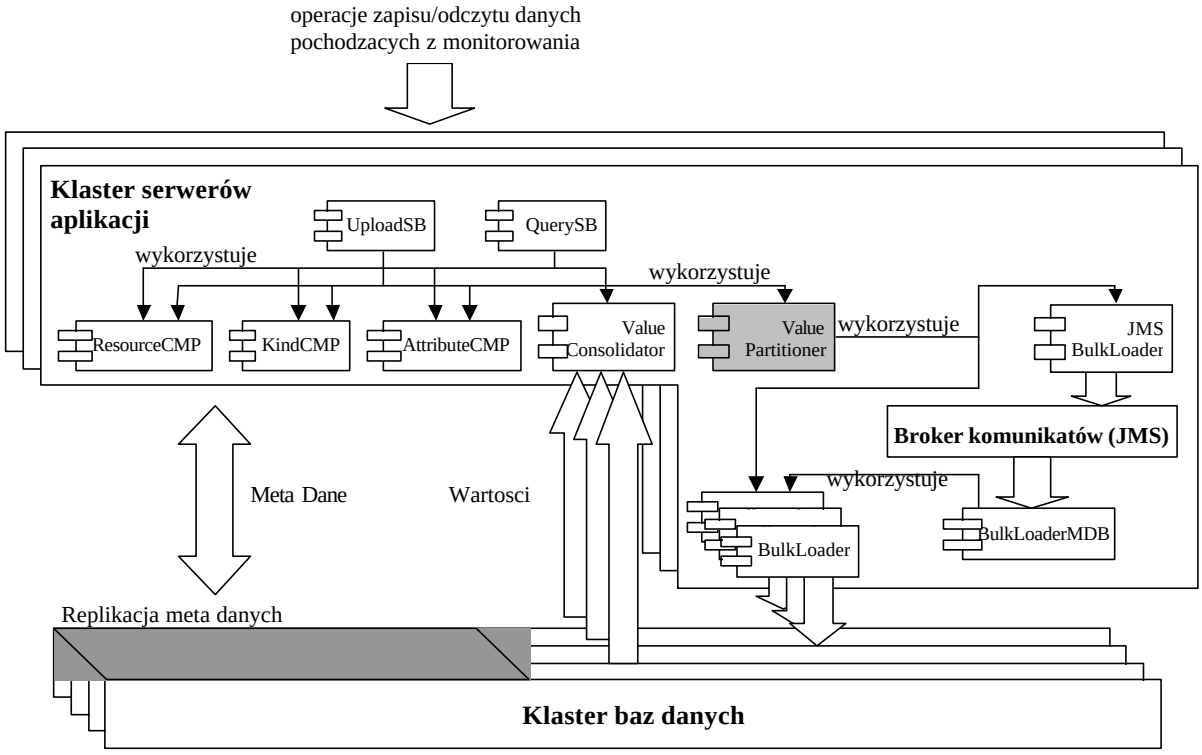
Utrzymanie wiezów integralnosci, w przypadku partycjonowanie tabeli danych, wymaga kontroli unikalnosci kluczy glównych pomiedzy partycjami. W SZiPD, dzieki zastosowaniu grupowych operacji zapisu i odczytu, indywidualizacja wartosci, której sluza klucze glówne, nie jest istotna. Beda one zatem generowane niezaleznie w kazdej instancji bazy danych.

4.4.4 Wady i zalety modelu

Najwieksza zaleta modelu ma byc zwiekszenie wydajnosci uzyskane dzieki partycjonowaniu danych; czy i w jakim stopniu zostalo ono osiagniete jest przedmiotem badan doswiadczalnych przedstawionych w rozdziale 5. Za zalete modelu nalezy tez uznac mozliwosc partycjonowania danych z wykorzystaniem heterogenicznych baz danych, co nie jest mozliwe w przypadku stosowania mechanizmów klasteryzacji i partycjonowania konkretnego silnika bazy danych. Wada rozwiazania jest duza komplikacja: potrzeba modyfikowania komponentu *QuerySB,* stworzenia dwóch nowych komponentów *ValuePartitioner* i *ValueConsolidator* oraz dosc zlozona konfiguracja. Dodatkowo system dziedziczy wade modelu ze sklastrowanym serwerem aplikacji, przedstawionego w punkcie 4.2; wywolania operacji zapisu sa wykonywane w sposób synchroniczny i nie moga byc kolejkowane, co powoduje brak odpornosci na chwilowe zwiekszenie wielkosci strumienia danych (piki w obciazeniu).

4.5 Model hybrydowy

Ostatnim z proponowanych modeli jest model hybrydowy – Rys. 37. Laczy on cechy modelu opartego na brokerze komunikatów oraz modelu opartego na partycjonowaniu danych.



Rys. 37 Hybrydowy model SZiPD.

Model hybrydowy wykorzystuje partycjonowanie danych oraz dwa tryby zapisu danych. Przy malym obciazeniu wykorzystywane jest partycjonowanie danych oraz synchroniczny tryb zapisu, który gwarantuje male opóznienia w zapisie danych. Sytuacja przeciazenia systemu powoduje aktywowanie brokera komunikatów, do którego przekazywana jest „nadwyzka" danych, która nie moze byc przetworzona w trybie synchronicznym.

Model wykorzystuje komponenty stworzone na potrzeby wczesniejszych rozwiazan. W trybie synchronicznym, dane zapisywane sa do róznych instancji baz danych przez komponenty *BulkLoader*. W trybie asynchronicznym komponent *JMSBulkLoader* przekazuje dane do brokera komunikatów, sa one nastepnie przetwarzane przez komponent *BulkLoaderMDB* i zapisywane w bazie danych przy wykorzystaniu komponentu *BulkLoader*. Odczyt danych jest realizowany podobnie jak w poprzednim modelu poprzez komponent *ValueConsolidator*.

Pierwotna koncepcja zakladala stworzenie modelu hybrydowego na bazie modelu wykorzystujacego broker komunikatów, rozszerzonego o mozliwosc zapisu danych w kilku

instancjach baz danych. Rozwiazanie takie mialoby, podobnie jak model oparty na brokerze komunikatów, wade w postaci wprowadzanego dodatkowego opóznienia zapisu danych, nawet w przypadku niskiego obciazenia systemu. Prezentowany w poprzednim punkcie model wykorzystujacy klaster bazy danych nie wprowadzal dodatkowych opóznien przy zapisie, nie radzil sobie jednak z chwilowym zwiekszaniem strumienia danych. Proponowany model hybrydowy laczy zalety obu rozwiazan, umozliwia zachowanie parametrów wydajnosciowych w sytuacji chwilowego przeciazenia systemu oraz niski czas zapisu, gdy system jest mniej obciazony. Kluczem do jego poprawnego dzialania jest wlasciwa detekcja momentu, w którym nalezy aktywowac broker komunikatów.

4.5.1 Wybór trybu asynchronicznego

Model hybrydowy zaklada dzialanie w trybie synchronicznym oraz wykorzystanie asynchronicznego trybu zapisu danych w sytuacji, w której zapis synchroniczny powodowalby spadek parametrów jakosciowych dzialania systemu: czasu odpowiedzi oraz poprawnosci wykonania ponizej ustalonych wartosci. Wykorzystanie brokera, czyli aktywacja asynchronicznego trybu zapisu, powinno byc zrealizowane zanim parametry spadna ponizej wartosci progowych. Rozwiazaniem mogloby byc monitorowanie aktualnego obciazenia systemu i baz danych oraz aktywowanie trybu asynchronicznego w przypadku zblizania sie do wartosci granicznych. Nie wszystkie bazy danych udostepniaja jednak informacje o swoim aktualnym obciazeniu, nie jest ona tez w zaden sposób standaryzowana i szybko sie dezaktualizuje, a jej pozyskiwanie i przetwarzanie wiaze sie z dodatkowym obciazeniem systemu, bazy i medium komunikacyjnego.

Aby zaprezentowac potencjal proponowanego modelu, autor proponuje rozwiazanie alternatywne, które bazuje na okresleniu ilosci polaczen z baza danych wykorzystywanych dla trybu synchronicznego. Przykladowo, jesli optymalna iloscia polaczen do bazy danych jest 50, 40 z nich mozna przeznaczyc dla trybu synchronicznego. Poniewaz tryb synchroniczny jest preferowany, w przypadku obciazenia powodujacego wykorzystanie mniej niz 40 polaczen zapis bedzie odbywal sie w sposób synchroniczny. Jesli nastapi zwiekszenie obciazenia (np. 100 równoczesnych zadan zapisu) to nadwyzka danych zostanie przekazana do brokera komunikatów i bedzie sukcesywnie przetwarzana przy pomocy pozostalych 10 polaczen. Realizacja powyzszego schematu dzialania, podobnie jak w modelu opisywanym w punkcie 4.4, jest oparta o wspóldzielona, cykliczna liste komponentów *BulkLoader*. Wykonanie operacji zapisu polega na pobraniu komponentu z listy, jego wykorzystaniu do

zapisania danych oraz zwolnieniu. Ilosc komponentów wykorzystywanych równoczesnie do zapisu synchronicznego jest kontrolowana i ograniczona wartoscia parametru *maxConcurrentSynchronousOperations*. Sytuacja, w której ilosc równoczesnie wykorzystywanych komponentów osiaga wartosc graniczna, oznacza potrzebe aktywowania trybu asynchronicznego, 'nadwyzka' danych jest przekazywana poprzez komponent *JMSBulkLoader* do brokera komunikatów. Do przetwarzania komunikatów z brokera komunikatów uzyty zostanie, opisany w punkcie 4.3.1, komponent *BulkLoaderMDB*.

4.5.2 Wady i zalety modelu

Koncepcja model hybrydowego zaklada wykorzystanie partycjonowania danych oraz róznych trybów zapisu: synchronicznego oraz opartego o broker komunikatów. Udzial poszczególnych trybów zapisu w ich calkowitej ilosci moze byc latwo zmieniany. W skrajnych przypadkach proponowane rozwiazanie moze dzialac wylacznie w jednym z trybów. Podejscie takie umozliwia adaptowanie sposobu dzialanie systemu do konkretnych warunków pracy. Na ile zaproponowane mechanizmy rzeczywiscie przyczyniaja sie do poprawy parametrów dzialania systemu oraz z jakim narzutem wiaze sie ich wprowadzenie, zostalo zweryfikowane poprzez przeprowadzenie odpowiednich testów wydajnosciowych przedstawionych w rozdziale 5.

4.6 Podsumowanie

Przedstawione w tym rozdziale modele SZiPD posiadaja z punktu widzenia uzytkownika te sama funkcjonalnosc – zbieraja i przechowuja dane pochodzace z monitorowania systemów rozproszonych – zgodnie z wymaganiami, które zawiera Tabela 8 przedstawiona w punkcie 3.6.4. Róznia sie jednak wlasnosciami niefunkcjonalnymi, które zbiera Tabela 9.

Model / **Wlasnosc**	**Model bazowy**	**Zoptymalizowany model bazowy**	**Model ze sklastro-wanym serwerem aplikacji**	**Model oparty na brokerze komunikatów**	**Model oparty na partycjono-waniu danych**	**Model hybrydowy**
Równowazenie obciazenia	–	–	✓	✓	✓	✓
Automatyczna reakcja na awarie serwera aplikacji	–	–	✓	✓	✓	✓

Automatyczna reakcja na awarie bazy danych	–	–	–	–	✓	✓
Wysoka dostepnosc	–	–	–	✓	✓	✓
Dynamiczne dodawanie instancji serwera aplikacji	–	–	✓	✓	✓	✓
Dynamiczne dodawanie instancji bazy danych	–	–	–	–	✓	✓
Wsparcie dla róznych baz danych	✓	✓	✓	✓	✓	✓
Odpornosc na chwilowe zwiekszenie obciazenia	–	?	?	?	?	?
Skalowalnosc	–	?	?	?	?	?

Tabela 9 Wybrane, niefunkcjonalne wlasnosci modeli SZiPD.

Powyzsze zestawienie jest oparte na teoretycznej analizie wlasnosci przedstawionych modeli. Zadaniem autora jest praktyczna weryfikacja postulowanych w tezie pracy wlasnosci systemu. Kazdy z prezentowanych modeli zostal zatem zaimplementowany, uruchomiony i przetestowany; uzyskane wyniki zostaly przedstawione w kolejnym rozdziale.

5 Testy skalowalnosci i wydajnosci SZiPD

„Things should be made as simple as possible, but not any simpler”[2]
Albert Einstein

Celem opisanych w tym rozdziale prac jest zbadanie wplywu rozszerzen, zaproponowanych dla bazowego modelu SZiPD, na uzyskiwana przez ten system wydajnosc i skalowalnosc, w zakresie:

- optymalizacji modelu bazowego, czyli wprowadzeniu komponentu realizujacego grupowa operacje zapisu,
- klasteryzacji serwera aplikacji,
- mechanizmów kolejkowych i przetwarzania asynchronicznego,
- mechanizmów partycjonowania danych,
- podejscia hybrydowego.

Przeprowadzone badania sluza okresleniu, na ile poprzez zwiekszanie zasobów sprzetowych systemu, mozna utrzymac jakosciowe parametry jego pracy pomimo zwiekszania przetwarzanego przez system strumienia danych.

Wydajnosc systemów komponentowych jest zazwyczaj oceniana i mierzona z punktu widzenia klientów tego typu systemów. System traktowany jest jak czarna skrzynka (ang. black box), dla której okreslane sa metryki bazujace na ilosci i czasie wykonania wywolywanych operacji. SZiPD jest testowany w podobnej konwencji, odpowiadajacej faktycznemu modelowi wykorzystania. Klientem SZiPD jest, stworzona na potrzeby testów, aplikacja zapisujaca meta-dane i dane.

Przestrzen parametrów konfiguracyjnych dla systemów komponentowych jest bardzo duza. SZiPD moze dzialac w oparciu o rózne elementy infrastruktury uruchomieniowej (ang. runtime environment): systemy operacyjne, oprogramowanie warstwy posredniej (wirtualna maszyna Java, serwery aplikacji, serwery kolejkowe) oraz bazy danych. Dla potrzeb eksperymentów zostaly one wybrane w sposób typowy dla wiekszosci spotykanych instalacji produkcyjnych, gdyz celem pracy nie jest ich porównywanie, a jedynie wykorzystanie dla potrzeb uruchomienia SZiPD. Jednak nawet takie zawezenie spektrum badan nie jest wystarczajace. Kazdy z elementów infrastruktury uruchomieniowej posiada szereg

[2] Rzeczy powinny byc tworzone tak prosto jak to mozliwe, ale ani odrobine prosciej.

parametrów konfiguracyjnych, których wartosc w bardzo istotny sposób wplywa na uzyskiwane wyniki wydajnosciowe. Aby móc dokonac porównania proponowanych rozwiazan, autor zdecydowal sie na wykonanie wstepnych testów, które pomogly w ustaleniu optymalnych parametrów dzialania elementów infrastruktury. Wszystkie opisane w rozdziale testy zostaly przeprowadzone w oparciu o te same parametry konfiguracyjne.

Przedstawione w niniejszym rozdziale badania stanowia praktyczna weryfikacje proponowanych rozwiazan pod wzgledem uzyskiwanych parametrów wydajnosciowych oraz skalowalnosci. Autor proponuje metodologie, srodowisko testowe, zestaw testów, metryki oraz kryteria ewaluacji, które sa podstawa dla przeprowadzenia eksperymentów obejmujacych rózne modele SZiPD w róznych konfiguracjach, poddawane zróznicowanemu obciazeniu. Sposród uzyskanych wyników wybrane, przedstawione i dyskutowane sa najbardziej znaczace. Rozdzial zamyka podsumowanie.

5.1 Przygotowanie testów

W tym punkcie przedstawiono notacje opisujaca konfiguracje testowanych modeli, sposoby generowania obciazenia oraz definicje metryk i parametrów wydajnosciowych.

5.1.1 Konfiguracje modeli

Zaproponowanym w rozdziale 4, kolejnym rozszerzeniom modelu bazowego zostaly nadane nastepujace, wykorzystywane w testach, symbole:

- M0 – model bazowy,
- M1 – zoptymalizowany model bazowy,
- M2 – model ze sklastrowanym serwerem aplikacji,
- M3 – model oparty na brokerze komunikatów,
- M4 – model oparty na partycjonowaniu danych,
- MH – model hybrydowy.

Uruchomienie powyzszych modeli wymaga wykorzystania elementów infrastruktury uruchomieniowej systemu; zostaly one oznaczone nastepujacymi skrótami: baza danych – „db”, broker komunikatów – „br” oraz serwer aplikacji – „as”. Elementy infrastruktury moga dzialac w oparciu o rózne zasoby sprzetowe, opis konfiguracji modelu zawiera informacje ile zasobów jest wykorzystywanych przez kazdy z elementów. Poniewaz wykorzystywane na

potrzeby testów zasoby sprzetowe sa homogeniczne, sposób ich przypisania do kazdej z warstw moze byc opisany ilosciowo, zgodnie z notacja:

$$Mx\{db=n, br=m, as=k\}$$

Mx - nazwa testowanego modelu,
n - liczba zasobów sprzetowych (serwerów fizycznych) przypisana bazom danych,
m - liczba zasobów sprzetowych (serwerów fizycznych) przypisana brokerom komunikatów,
k - liczba zasobów sprzetowych (serwerów fizycznych) przypisana serwerom aplikacji.

Testy prowadzone sa na serwerach jednoprocesorowych. Przyjeto zatem, ze na jednym fizycznym serwerze dziala jedna instancja elementu infrastruktury modelu (baza danych, serwer aplikacji, broker). Poniewaz mozliwe jest laczenie róznych elementów infrastruktury w jednym procesie (serwer aplikacji i broker komunikatów), a takze ich uruchamianie w ramach róznych procesów dzialajacych na tym samym fizycznym serwerze (broker komunikatów i baza danych), dlatego notacja zostala rozszerzona o mozliwosc specyfikowania liczby współdzielonych zasobów:

$$Mx\{db+br=n, as=m\}, Mx\{db=k, as+br=l\}$$

Mx - nazwa testowanego modelu,
n - liczba zasobów sprzetowych współdzielonych przez bazy danych i broker komunikatów,
m - liczba zasobów sprzetowych przypisana serwerom aplikacji,
k - liczba zasobów sprzetowych przypisana bazom danych,
l - liczba zasobów sprzetowych współdzielonych przez serwer aplikacji i broker komunikatów.

Zapis przykladowych konfiguracji modeli w proponowanej notacji i jej znaczenie:

- M1{db=1, as=1} – zoptymalizowany model bazowy dzialajacy w oparciu o dwa serwery; baze danych na jednym i serwer aplikacji dzialajacy na drugim serwerze,
- M3{db=1, br=2, as=3} – model oparty na brokerze komunikatów dzialajacy w oparciu o szesc serwerów, baze danych na jednym serwerze, broker komunikatów na dwóch serwerach oraz serwer aplikacji na trzech serwerach,
- MH{db=1, br+as=3} – model hybrydowy dzialajacy w oparciu o cztery serwery, baze danych na jednym serwerze oraz broker komunikatów i serwer aplikacji dzialajace jednoczesnie na trzech, współdzielonych serwerach.

W celu uproszczenia zapisu w dalszej czesci, wprowadzono pojecie krotnosci konfiguracji modelu, której wartosc jest iloscia fizycznych serwerów na jakich model jest uruchamiany.

$$|Mx\ \{db=n, br=m, as=k\}| = n+m+k$$

|Mx{..} | - krotnosc modelu Mx w okreslonej konfiguracji testowej,
Mx - nazwa testowanego modelu,
n - liczba zasobów sprzetowych (serwerów fizycznych) przypisana bazom danych,

m - liczba zasobów sprzetowych (serwerów fizycznych) przypisana brokerom komunikatów,
k - liczba zasobów sprzetowych (serwerów fizycznych) przypisana serwerom aplikacji.

Porównywanie uzyskanych wyników wydajnosciowych róznych modeli ma sens wylacznie pomiedzy konfiguracjami o tej samej krotnosci.

5.1.2 Sposób generowanie obciazenia

Testy wydajnosciowe zostaly przeprowadzone w oparciu o oprogramowanie Grinder [Gri05], umozliwiajace symulowanie obciazenia systemu oraz wspierajace rejestracje wartosci metryk. Oprogramowanie to sklada sie z trzech elementów:

- procesów roboczych (ang. worker node) – uruchamianych na potrzeby konkretnego testu procesów, wykonujacych okreslone operacje testowe przy uzyciu pewnej liczby watków,
- agentów – uruchamianych na kazdym z komputerów majacych generowac testowe obciazenie procesów, które kontroluja procesy robocze oraz odpowiadaja za przekazywanie wyników z danej maszyny do konsoli,
- konsoli zarzadzajacej – posiadajacego graficzny interfejs uzytkownika programu, odpowiadajacego za koordynowanie agentów i procesów roboczych, uruchamianie testów oraz zbieranie i prezentacje wyników.

Dla kazdego agenta ustalana jest liczba procesów roboczych oraz liczba watków dzialajacych w ramach jednego procesu roboczego. Kazdy watek reprezentuje klienta systemu. Zachowanie symulowanego klienta, jest opisane skryptem w jezyku Jyhton i polega na wywolaniu na SZiPD nastepujacych typowych operacji:

- zarejestrowanie w systemie nowego zasobu,
- zdefiniowanie dla tego zasobu zbioru monitorowanych atrybutów, zawierajacego jeden atrybut prosty, jeden atrybut strukturalny oraz dwa, rozszerzajac go, atrybuty proste o odpowiednio o typach: napis, liczba i liczba zmiennoprzecinkowa, stanowiace meta-dane
- periodyczne przekazywanie do systemu danych w paczkach o okreslonej wielkosci i równej ilosci wartosci kazdego ze zdefiniowanych atrybutów.

Waznym elementem symulowania dzialania klientów jest ustalenie charakterystyki strumienia danych jaki bedzie generowany. Wykazanie badanych w ramach testów SZiPD wlasnosci, wymaga mozliwosci generowania obciazenia jednostajnego oraz, dla niektórych scenariuszy

testowych, pików w obciazeniu. Prawidlowe porównanie modeli wymaga zdolnosci do generowania strumienia w sposób powtarzalny, tak by poszczególne modele byly poddawane obciazeniu o takiej samej charakterystyce. Wymaga to wprowadzenia dodatkowych parametrów opisujacych dzialanie aplikacji testowej:

- **odstepu pomiedzy wywolaniami operacji** (ang. operation execution delay) – czyli czasu, co jaki klient bedzie rozpoczynal operacje zapisu danych do systemu. Wymagalo to modyfikacji standardowego dzialania systemu Grinder,
- **maksymalnego opóznienia startu** (ang. start delay) – uzyskanie jednostajnego obciazenia systemu wymaga rozsynchronizowania momentu rozpoczecia zapisu danych przez klientów, bedzie ono uzyskane poprzez opóznianie rozpoczecia dzialania klienta o losowa wartosc z przedzialu [0, maksymalne opóznienie startu]. Ustawienie wartosci maksymalnego opóznienia startu na wartosc równa odstepowi pomiedzy wywolaniami operacji, spowoduje uzyskanie mozliwie jednostajnego obciazenia. Strumien danych uzyskany poprzez ustawienie wartosci maksymalnego opóznienia startu na 0 bedzie zawieral duze chwilowe wzrosty obciazenia (piki).

Poniewaz konfiguracje wszystkich komputerów, na których dzialaja klienci sa takie same, do opisu symulowanego obciazenia systemu wystarczajaca jest nastepujaca notacja:

(X*Y*Z, A, B)

X – liczba zasobów sprzetowych, fizycznych komputerów generujacych obciazenie,
Y – liczba procesów roboczych na kazdym z komputerów,
Z – liczba watków w ramach kazdego procesu roboczego,
A – okres czasu pomiedzy kolejnym operacji zapisu danych w sekundach,
B – maksymalna wartosc opóznienia rozpoczecia dzialania w sekundach.

Iloczyn X,Y,Z, a wiec ilosci komputerów generujacych obciazenie, procesów roboczych oraz ilosci watków odpowiada liczbie klientów.

5.1.3 Notacja opisu konfiguracji testowych

Kompletna konfiguracja dla przeprowadzenia testu sklada sie z konfiguracji modelu oraz konfiguracji klientów generujacych obciazenie systemu. Zostala ona nazywana konfiguracja testowa i jest opisywana przy uzyciu zapisanych lacznie notacji dla obu typów konfiguracji.

Przykladowy zapis konfiguracji testowych w proponowanej notacji ma nastepujaca postac:

- M0{as+db=1}(5*3*5, 10, 10) – model bazowy dzialajacy w oparciu o jeden serwer, na którym uruchomiona jest baza danych i serwer aplikacji obciazany przez 75

klientów, dzialajacych na pieciu komputerach, na których uruchomiono po trzy procesy robocze posiadajace po piec watków kazdy, kazdy z watków zapisuje paczke danych raz na 10 sekund, maksymalne opóznienie startu wynosi równiez 10 sekund co powoduje uzyskanie jednostajnego obciazenia systemu.

- M3{db=1, br=2, as=3}(5*5*10, 5, 0) – model oparty na brokerze komunikatów dzialajacy w oparciu o szesc serwerów, baze danych na jednym serwerze, broker komunikatów na dwóch serwerach oraz serwer aplikacji na trzech serwerach, obciazany przez 250 klientów, dzialajacych na pieciu komputerach, na których uruchomiono po piec procesów roboczych posiadajacych po dziesiec watków kazdy, kazdy z watków zapisuje paczke danych co 5 sekund, maksymalne opóznienie startu wynoszace 0, powoduje równoczesne rozpoczecie dzialania wszystkich watków, co skutkuje powstawaniem duzych chwilowych wzrostów obciazenia systemu.

5.1.4 Metryki

Przeprowadzenie testów wymaga zdefiniowania metryk, które beda mierzone w czasie testów oraz okreslenia sposobu ich wyznaczania. Dla potrzeb testów SZiPD okreslono cztery metryki. Sa one definiowane w kontekscie operacji, a wiec pojedynczego, zdalnego wywolania metody, wykonywanego na poddawanym testom systemie, przez zdalna aplikacje testujaca (klienta). Metryki sa mierzone dla konfiguracji testowych objetych testami, a wiec dla róznych konfiguracji modeli poddawanych róznemu obciazeniu.

Czas odpowiedzi systemu (SRT)

Czas odpowiedzi systemu SRT (ang. system response time) jest, mierzonym w milisekundach, czasem trwania wywolania operacji, liczonym po stronie klienta jako róznica czasu poczatku wywolania i czasu jego zakonczenia.

$$SRT_o = te_o - ts_o$$

STR_o - czas odpowiedzi systemu dla operacji o,
ts_o - chwila czasu rozpoczecia wykonywania przez klienta operacji o,
te_o - chwila czasu zakonczenia wykonywania przez klienta operacji o.

Pomiar realizowany jest przez oprogramowanie Grinder, które dla kazdej wykonywanej w ramach testu operacji mierzy SRT.

Ilosc transakcji na sekunde (TPS)

Ilosc transakcji na sekunde TPS (ang. transactions per second) jest wyliczana jako suma ilosci operacji zakonczonych w kazdym z jednosekundowych przedzialów, na jakie zostal podzielony czas trwania testu.

$$TPS_i = a_{(ti:\ ti+1)}$$

TPS_i – ilosc transakcji na sekunde dla i-tego przedzialu czasu,
$a_{(ti:\ ti+1)}$ – ilosc operacji, których wykonywanie zostalo zakonczone w przedziale czasu ti do ti +1,
ti – chwila czasu oznaczajaca poczatek i-tego przedzialu czasowego.

Pomiar realizowany jest przez oprogramowanie Grinder, które zapisuje czas rozpoczecia i dlugosc trwania kazdej operacji testowej. Ilosc wszystkich operacji zakonczonych w i-tym przedziale czasu stanowi wartosc TPS i-tego przedzialu.

Przepustowosc systemu na sekunde (DTPS)

Przepustowosc systemu na sekunde DTPS (ang. data throughput per second) jest wartoscia oznaczajaca ilosc danych jaka system przyjmuje do przetworzenia w ciagu sekundy. Jest wyliczana dla kazdego przedzialu czasu jako suma ilosci danych zapisanych w kazdej operacji zakonczonej w tym przedziale.

$$DTPS_i = \sum_{o_t,\ t \in (ti:\ ti+1)} d_{o_t}$$

$DTPS_i$ – przepustowosc systemu na sekunde dla i-tego przedzialu czasu,
ti – chwila czasu oznaczajaca poczatek i-tego przedzialu czasowego,
d_o – ilosc danych zapisywana w operacji o,
$o_{t\ t \in (ti:\ ti+1)}$ – operacja zakonczona w przedziale czasu ti do ti +1.

W przypadku operacji synchronicznych $DTPS_i$ oznacza zarówno liczbe danych przyjetych do przetworzenia, jak i danych faktycznie przetworzonych. Dla operacji asynchronicznych wartosc $DTPS_i$ oznacza jedynie ilosc danych przyjetych przez system do przetworzenia, ilosc danych faktycznie przetworzonych, czyli zapisanych w bazie danych moze byc mniejsza. Wyznaczanie $DTPS_i$ polega na mnozeniu ilosci transakcji, zakonczonych w kazdym przedziale czasu – TSP_i przez wielkosc paczki danych.

Stopa bledów

Stop bledów testu ERR (ang. error rate) oznacza procentowy udzial odrzuconych lub zakonczonych wyjatkiem operacji zapisu danych w stosunku do wszystkich operacji zapisu danych wykonywanych w ramach testu.

$$ERR = errT / allT * 100\%$$

ERR - stopa bledu testu,
errT - ilosc operacji odrzuconych lub zakonczonych wyjatkiem,
allT - ilosc wszystkich operacji wykonanych w ramach testu.

Bledy w dzialaniu systemu moga sie pojawiac np. w sytuacji przeciazenia systemu czy zbyt dlugiego czasu trwania operacji zapisu, na skutek przekroczenia maksymalnych czasów wywolania zdalnej operacji oraz maksymalnego czasu trwania transakcji. Wskaznik ERR jest wyliczany na podstawie danych dostarczonych przez oprogramowanie Grinder.

5.1.5 Parametry wydajnosciowe i jakosciowe

Na podstawie przedstawionych w poprzednim punkcie metryk, okreslono parametry wydajnosciowe i jakosciowe charakteryzujace dzialanie systemu. Obejmuja one wartosci minimalna, maksymalna, srednia oraz odchylenie standardowe metryk, obliczane dla okresu czasu objetego testem. Parametry te sa obliczane wedlug ponizszego, ogólnego wzoru:

$$\forall_{X \in \{SRT, TPS, DTPS\}} \begin{cases} \text{minX} = \min(X_i),\ i \in [1, n] \\ \text{maxX} = \max(X_i),\ i \in [1, n] \\ \text{avgX} = \dfrac{\sum_{i=1}^{n} X_i}{n} \\ \text{devX} = \sqrt{\dfrac{\sum_{i=1}^{n} (X_i - avgX)^2}{n}} \end{cases}$$

X_i – wartosc metryki X dla i-tego przedzialu czasowego,
n – ilosc przedzialów czasowych.

Parametry wydajnosciowe systemu obejmuja nastepujace wartosci:

- minTPS, maxTPS, avgTPS, devTPS,
- minDTPS, maxDTPS, avgDTPS, devDTPS.

Zgodnie z zaproponowana w punkcie 2.4 definicja wydajnosci systemu komponentowego, bedzie ona wyznaczana dla okreslonych parametrów jakosci dzialania systemu, wybranych sposród nastepujacych:

- minSRT, maxSRT, avgSRT, devSRT,
- ERR.

5.2 Srodowisko testowe

Wlasciwy wybór i konfiguracja srodowiska testowego ma kluczowe znaczenie dla poprawnosci przeprowadzanych eksperymentów i otrzymanych wyników. Dobrze dobrane srodowisko testowe dla aplikacji rozproszonych powinno zapewniac stabilnosc, separacje kanalów komunikacyjnych od ruchu lokalnego, elastycznosc generowania obciazenia oraz kontrole nad zachowaniem klientów, dodatkowo, w przypadku testowania wielu róznych konfiguracji, mozliwosc szybkiej i latwej rekonfiguracji. Najwazniejszym parametrem warunkujacym przeprowadzenie testów pozostaje jednak dostepnosc zasobów i to zarówno sprzetowych jak i oprogramowania.

5.2.1 Infrastruktura sprzetowa

Testy zostaly uruchomione na platformie SUN Fire B1600 Blade; dostepna w Katedrze Informatyki AGH instalacja zawiera 48 komputerów SUN Fire B100s Blade Server – Rys. 38. Na potrzeby testów zostalo wydzielone 16 komputerów, tworzacych dedykowany klaster, calkowicie niezalezny od pozostalych komputerów platformy oraz izolowany dzieki zastosowaniu technologii wirtualnych sieci prywatnych (VPN) oraz prywatnej adresacji sieciowej.

Rys. 38 Farma 48 komputerów SUN Fire 100s Blade Server.

Parametry sprzetowe wszystkich wykorzystywanych komputerów SUN Fire sa jednakowe; Tabela 10 zbiera ich podstawowe parametry.

Parametr	**Wartosc**
Procesor	650MHz Ultra SPARC Iii
Pamiec	1 GB PC133 DIMM
Dysk	30GB, Ultra ATA 100, 5400rpm
Siec	1000-Mbps Ethernet
System operacyjny	SUN Solaris 9

Tabela 10 Konfiguracja komputerów SUN Fire 100s Blade Server.

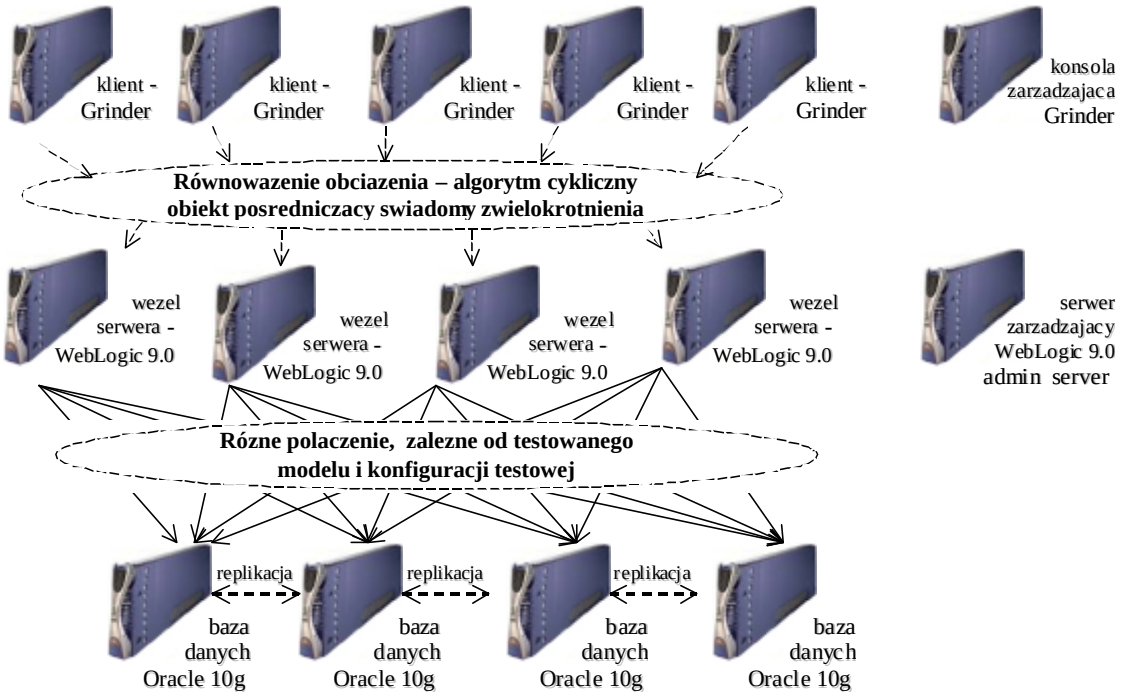
5.2.2 Infrastruktura programowa i konfiguracja

Sposród dostepnych serwerów aplikacji autor wybral WebLogic Application Server 9.0 jako produkt najbardziej uznany w srodowisku komercyjnym. Wersja i parametry wirtualnej maszyny Java zostaly dobrane zgodnie z zaleceniami producenta serwera aplikacji. Bardzo wysoka wydajnosc, mozliwosci klastrowania oraz posiadana licencja sklonily do wyboru bazy danych Oracle 10g jako motoru bazy danych dla przeprowadzanych testów. W testach wykorzystano równiez broker komunikatów dostarczany z serwerem aplikacji WebLogic.

Typ oprogramowania	**Produkt**
Serwer Aplikacji J2EE	BEA WebLogic 9.0
Wirtualna maszyna Java	SUN JDK 1.5.04
Baza danych	Oracle 10g
Broker komunikatów	BEA WebLogic 9.0 JMS

Tabela 11 Wersje oprogramowania wykorzystywanego w testach

Sposób rozmieszczenia poszczególnych produktów na klastrze obrazuje Rys. 39.



Rys. 39 Architektura srodowiska testowego.

Konfiguracja tak przygotowanego srodowiska obejmowala:

- konfiguracje farmy serwerów Blade, systemu operacyjnego Solaris 9 oraz sieci,
- stworzenie farmy serwerów aplikacji, zdefiniowanie klastra,
- definiowanie zródel danych i pul polaczen z baza danych,

- aktywacja i konfiguracja brokera komunikatów oraz magazynu danych dla komunikatów (ang. persistent data store) opartego na lokalnym systemie plików,
- stworzenie baz danych oraz ustalenie mechanizmów replikacji multi-master [ORA05]
- strojenie systemu.

Wiekszosc prac konfiguracyjnych zostala realizowana poprzez dostepne przez przegladarke WWW graficzne interfejsy konsoli zarzadzajacych serwera administracyjnego WebLogic 9.0 oraz bazy danych Oracle.

5.3 Metodyka testów

Pierwotnie zaplanowane i wykonane testy SZiPD bazowaly na koncepcji testów obciazeniowych (ang. stress testing), czyli obciazania systemu bardzo duza iloscia operacji oraz mierzeniu ilosci operacji wykonanych przez system w jednostce czasu (TPS). Uzyskane w wyniku przeprowadzonych testów rezultaty pokazywaly jednak wylacznie charakterystyke modeli poddawanych ekstremalnie duzemu obciazeniu, z której nie mozna poprawnie wnioskowac o ich zachowaniu w trakcie ‘normalnej’ eksploatacji. Ponadto charakteryzowaly sie one bardzo duza wariancja, wynikajaca glównie z niepowtarzalnosci generowanego obciazenia. Bledem zastosowanej metodologii bylo tez wylacznie ilosciowe porównywanie przepustowosci systemu, podczas gdy w praktyce jakosc przetwarzania ma równie wazne, jesli nie istotniejsze znaczenie. Sytuacja ta zmusila autora do poszukiwania innej metodologii.

Do badania parametrów wydajnosciowych systemu zaproponowana zostala koncepcja, bazujaca na pojeciu punktu pracy systemu.

> **Punkt pracy systemu** jest taka konfiguracja symulowanego obciazenia konkretnej konfiguracji testowej (a wiec modelu i elementów infrastruktury), przy której osiagane sa maksymalne wartosci parametrów wydajnosciowych oraz zachowane zadane parametry jakosciowe.

Porównanie poszczególnych modeli i konfiguracji polega na porównaniu parametrów wydajnosciowych uzyskanych w punktach pracy, a wiec najlepszych, sposród uzyskanych przez modele w konkretnej konfiguracji wyników, przy których zachowane byly zadane parametry wydajnosciowe. Takie podejscie wymagalo:

- Okreslenia wymagan uzytkownika, czyli ustalenia wartosci parametrów jakosciowych, których spelnienie bylo warunkiem uznania przeprowadzonego testu za poprawny,
- Zmodyfikowania sposobu generowania obciazenia systemu tak, by mozna nim bylo latwo sterowac oraz aby zapewnic jego powtarzalnosc, pomimo róznej dlugosci czasu trwania wykonywanych operacji,
- Mierzenia zarówno parametrów wydajnosciowych jak i jakosciowych.

Wyznaczenie punktu pracy bazowalo wprost na definicji i bylo realizowane poprzez wykonywanie testów poddajacych badana konfiguracje coraz wiekszemu obciazeniu, generowanemu poprzez zwiekszanie ilosci symulowanych klientów systemu, az do momentu, w którym przekroczone zostaly dopuszczalne wartosci parametrów jakosciowych. Konieczne bylo wielokrotne wykonywanie testów konkretnego modelu w okreslonej konfiguracji, poddawanemu róznemu obciazeniu. Sposród uzyskanego zbioru wyników, wybierany byl jeden o najlepszych parametrach wydajnosciowych i spelniajacy zadane kryterium jakosciowe. Wartosci uzyskane w punkach pracy róznych modeli i w róznych konfiguracjach sa przedmiotem porównania i analizy zawartych w punkcie 5.4.

Wyznaczanie punktów pracy jest podejsciem zgodnym z przyjeta w punkcie 2.4 definicja wydajnosci systemu, daje mozliwosc pokazania istotnych róznice pomiedzy modelami, jest jednak dosc pracochlonne. Opracowanie i praktyczna weryfikacja uzytecznosci takiej metody do testowania wlasnosci systemów komponentowych jest jednym z dokonan pracy.

W kolejnych punktach przedstawiony zostal szczególowy przebieg testu, okreslone parametry uruchomieniowe dla testu oraz wymagania przyjete dla parametrów jakosciowych.

5.3.1 Przebieg testów

Kazdy z testów przebiegal wedlug tego samego scenariusza, kazdy posiadal takie same warunki poczatkowe: pusta baza danych, nowo uruchomione instancje wszystkich wykorzystywanych programów (baza danych, serwer aplikacji, broker). Dla wszystkich testów wykorzystano ten sam skrypt opisujacy dzialanie klienta. Testy polegaly na zestawieniu wybranej konfiguracji oraz uruchomieniu oprogramowania Grinder, symulujacego dzialanie odpowiedniej ilosci klientów, na czas 320 sekund. Zgodnie z zaleceniami zawartymi w [Gri05], zbieranie danych bylo przesuniete w czasie w stosunku do rozpoczecia dzialania przez klientów i rozpoczynalo sie 10 sekund po ich uruchomieniu.

Zbieranie statystyk rozpoczynalo sie w 10 sekundzie i trwalo do 310. sekundy. Kazdy z testów byl powtarzany trzykrotnie, a uzyskane metryki przeliczane zgodnie z definicjami parametrów wydajnosciowych okreslonymi w punkcie 5.1.5.

5.3.2 Okreslenie parametrów uruchomieniowych

Przeprowadzenie testów wymagalo ustalenia wartosci wielu parametrów konfiguracyjnych serwerów aplikacji, brokera komunikatów, bazy danych oraz samych testów. W tym celu przeprowadzone zostaly testy wstepne w konfiguracji M1{db=1, as=1}(1*1*5, 5, 5). Wybrane parametry zostaly – Tabela 12; sa one stosowane do przeprowadzania wiekszosci testów, ewentualne zmiany sa kazdorazowo zaznaczone.

Nazwa	**Opis**	**Wartosc**
Wielkosc paczki danych	Wielkosc paczki danych przesylana przez klienta ma stala wielkosc.	25
Ilosc polaczen do jednej instancji bazy danych	Ilosc otwartych polaczen zdefiniowanych w pulach polaczen kazdego serwera aplikacji dla kazdej bazy danych. Wartosc poczatkowa oraz maksymalna.	15, 100
Ilosc komponentów SessionBean	Ilosc komponentów sesyjnych w puli serwera aplikacji, dla kazdego komponentu. Wartosc poczatkowa oraz maksymalna.	40, 100
Ilosc komponentów MDB	Ilosc komponentów sterowanych komunikatami w puli serwera aplikacji, dla kazdego komponentu. Wartosc poczatkowa oraz maksymalna.	2, 5

Tabela 12 Wybrane wartosci parametrów konfiguracji testowanych.

5.3.3 Wymagane parametry jakosciowe

We wszystkich przeprowadzonych testach przyjeto te same wymagania odnosnie jakosci dzialania systemu. Jesli dla konkretnego modelu testowanego w okreslonej konfiguracji testowej przekroczone zostaly maksymalne wartosci dopuszczalne, chocby jednego z parametrów, to model ten byl uznawany za nie dzialajacy w okreslonej konfiguracji, co dyskwalifikowalo uzyskane przez niego wyniki wydajnosciowe. Graniczne wartosci parametrów wydajnosciowych zbiera Tabela 13.

Nazwa	**Opis**	**Maksymalna dopuszczalna wartosc**
ERR	Stopa blednie wykonanych operacji	0%
avgSRT	Sredni czas odpowiedzi systemu	0,5 s
devSRT	Odchylenie standardowe odpowiedzi systemu	0,5 s

Tabela 13 Wymagane parametry jakosciowe systemu.

W ramach badan wydajnosci i skalowalnosci SZiPD przeprowadzono lacznie kilkaset testów, które obejmowaly wszystkie szesc modeli w róznych konfiguracjach, poddawanych róznym obciazeniom. Uzyskane wyniki zostaly zebrane w kolejnym punkcie.

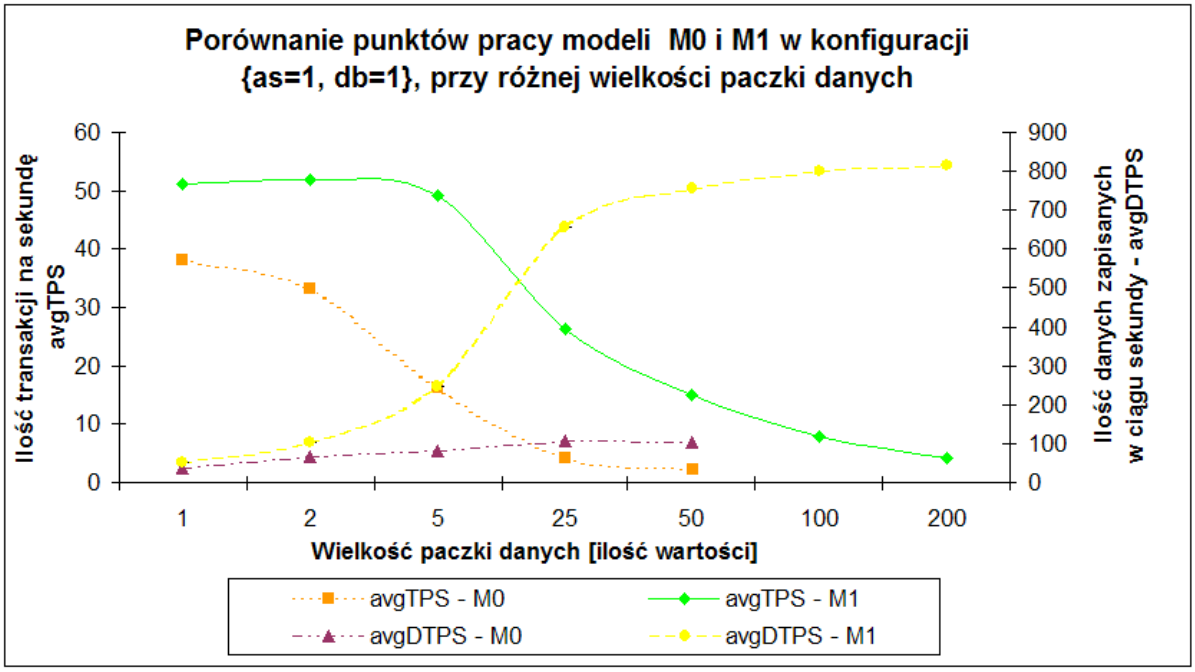
5.4 Uzyskane wyniki

W tym punkcie zawarto wybrane wyniki przeprowadzonych badan eksperymentalnych. Sposród przeprowadzonych testów, autor wybral, zilustrowal oraz omówil te rezultaty, które najlepiej oddaja badane wlasnosci poszczególnych modeli oraz sluza wykazaniu prawdziwosci tej czesci tezy pracy, która mówi o zapewnieniu odpowiedniej wydajnosci i skalowalnosci.

W kolejnych punktach przedstawione zostalo: porównanie modeli M0 i M1 weryfikujace wzrost wydajnosci uzyskany na skutek zastosowania brokera komunikatów, wplyw klasteryzacji serwera aplikacji na wzrost wydajnosci systemu oraz (w kolejnym punkcie) zalety modelu M3, wynikajace z wykorzystania brokera komunikatów. Nastepnie przedstawiono wyniki badan skalowalnosci modeli M4 i MH oraz ich odpornosci na chwilowy wzrost wielkosci strumienia danych. Podsumowanie uzyskanych wyników przedstawione jest w punkcie 5.5.

5.4.1 Wplyw optymalizacji modelu bazowego

Celem testu jest praktyczna weryfikacja stopnia poprawy wydajnosci modelu bazowego, uzyskana poprzez jego modyfikacje, zaproponowana w rozdziale 4.1. Model bazowy M0 oraz model zoptymalizowany M1 zostaly uruchomione w oparciu o te sama konfiguracje sprzetowa {as=1, db=1}. W celu wyznaczenia punktów pracy systemów, modele byly obciazane strumieniem danych o stalej wielkosci. Wielkosc strumienia byla w kolejnych testach zwiekszana, poprzez zwiekszanie ilosci watków obciazajacych system o jeden, az do przekroczenia zadanych parametrów jakosciowych. Najwieksza sposród uzyskanych wartosci avgTPS, przy której spelnione zostala zamieszczona na wykresie. Eksperymenty byly przeprowadzone kolejno dla kazdej badanej wielkosci paczki danych. Uzyskane wyniki zostaly zaprezentowane na Rys. 40.



Rys. 40 Porównanie punktów pracy modeli M0 i M1 w konfiguracji {as=1, db=1}, przy róznej wielkosci paczki danych.

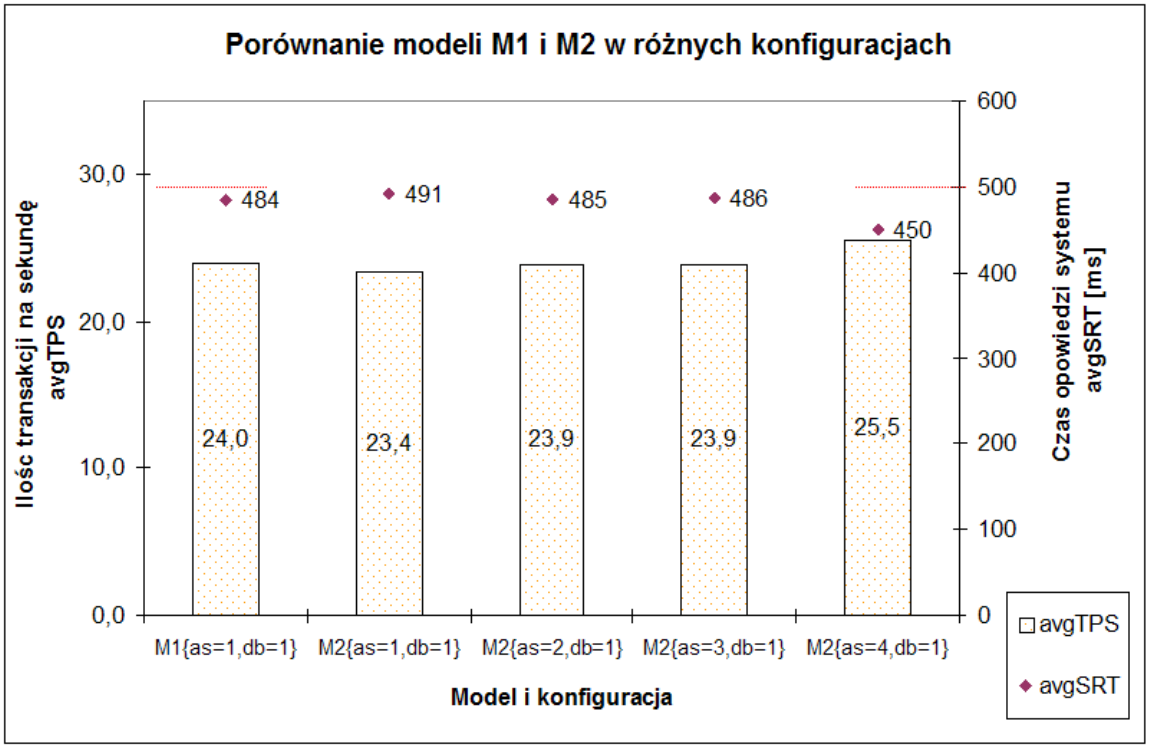
Na Rys. 40 pokazano srednie wartosci TPS i DTPS punktów pracy wyznaczonych niezaleznie dla kazdej wielkosci paczki obu badanych modeli. Nalezy zauwazyc, ze dla malej wielkosci paczki danych, uzyskane przez modele wartosci nie róznia sie w sposób znaczacy. Przy wiekszym rozmiarze paczki danych, w modelu M0 srednia ilosc transakcji (avgTPS) maleje. Zwiekszenie rozmiaru paczki powyzej 50 wartosci powoduje, ze model nie spelnia wymagan jakosciowych (sredni czas odpowiedzi systemu powyzej 0,5 s.), stad brak wartosci na wykresie. Model zoptymalizowany M1, wykorzystuje komponent do zapisu grupowego. Dla paczek o malej wielkosci jego parametry wydajnosciowe sa porównywalne z modelem bazowym, przy wiekszym rozmiarze paczki danych sa one o ponad rzad wielkosci lepsze.

Przeprowadzony test wskazuje na duzy wzrost wydajnosci systemu uzyskany dzieki optymalizacji modelu bazowego. Potwierdza to slusznosc wykorzystania zaproponowanej w punkcie 4.1.1 koncepcji komponentu dla grupowej operacji zapisu.

5.4.2 Wplyw klasteryzacji na zwiekszenie wydajnosci systemu

Celem testu jest sprawdzenie na ile klastrowanie serwera aplikacji podnosi wydajnosc systemu. W tym celu dokonano porównania modelu M1 oraz modelu M2 dzialajacego w róznych konfiguracjach sprzetowych: {as=1,db=1}, {as=2,db=1}, {as=3,db=1} oraz {as=4,db=1}. Wielkosc paczki danych wynosi 25 wartosci i jest, podobnie jak pozostale parametry konfiguracyjne, zgodna z przyjetymi dla testów wartosciami, które zawiera Tabela

12. System byl obciazany strumieniem danych o stalej wielkosci. Uzyskane wyniki zostaly zaprezentowane na Rys. 41.



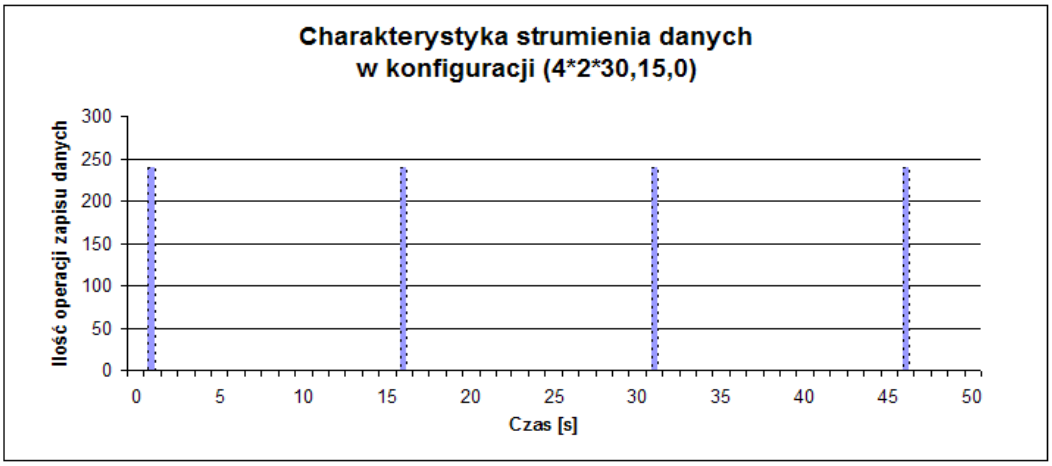
Rys. 41 Porównanie modeli M1 i M2 w róznych konfiguracjach.

Zawarte na Rys. 41 dane obejmuja srednia ilosc transakcji na sekunde (avgTPS) oraz sredni czas odpowiedzi systemu (avgSRT) dla punktów pracy wyznaczonych dla modelu M1 i M2 w kolejnych testowanych konfiguracjach. Przerywana linia na poziomie 500 ms. oznacza graniczna, dopuszczalna wartosc avgSRT. Nalezy zauwazyc, ze uzyskane przez oba modele we wszystkich konfiguracjach wartosci avgTPS sa zblizone. Fakt ten prowadzi do dwóch wniosków. Po pierwsze na podstawie porównania M1{as1,db-1} i M2{as=1,db=1}, mozna stwierdzic ze model M2 nie wnosi istotnych narzutów w stosunku do modelu M1. Drugim, wazniejszym wnioskiem jest stwierdzenie, ze klasteryzacja samego serwera aplikacji jest dzialaniem niewystarczajacym do poprawy parametrów wydajnosciowych SZiPD. W zwiazku z tym, celowe jest uzycie innych technik umozliwiajacych uzyskanie wlasnosci skalowalnosci, które zostaly wykorzystane w kolejnych modelach.

5.4.3 Odpornosc modeli M2 i M3 na chwilowy wzrost wielkosci strumienia danych

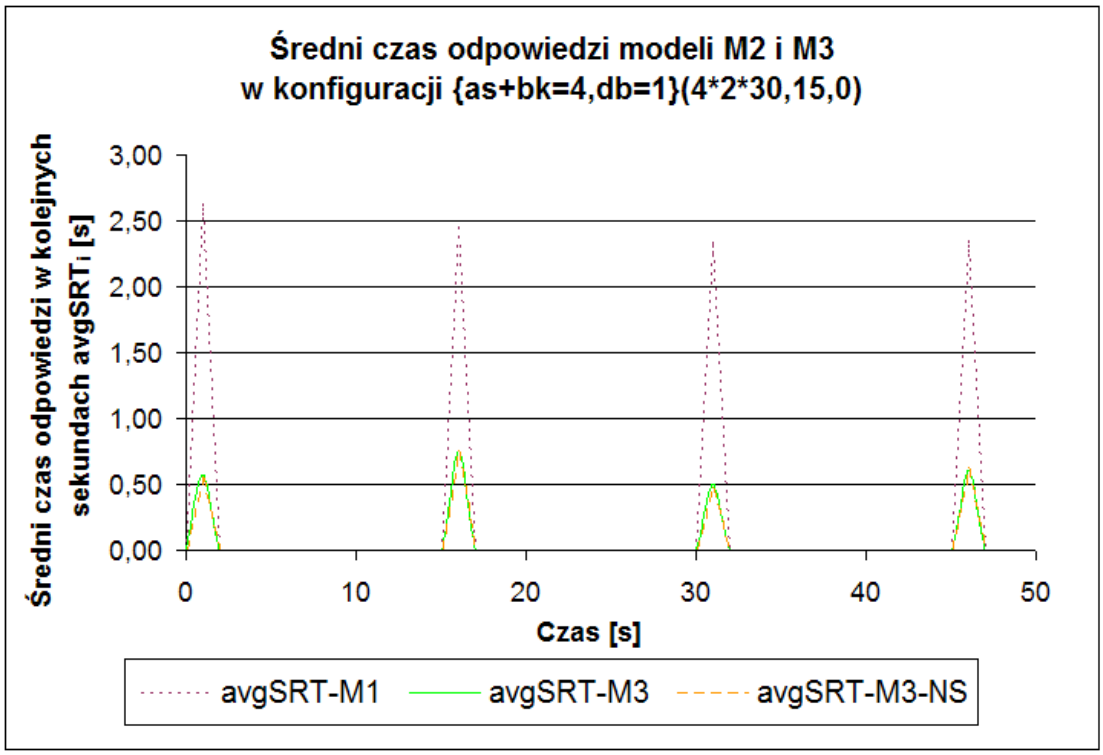
Celem testu jest sprawdzenie, jaki wplyw na zachowanie parametrów jakosciowych (avgSRT, ERR) ma chwilowy, duzy wzrost strumienia danych. Porównane zostaly dzialajace na klastrze serwerów aplikacji modele M2 i M3, a wiec odpowiednio model synchronicznie

zapisujacy dane do bazy danych oraz model wykorzystujacy broker komunikatów. Zadany strumien danych zostal uzyskany poprzez odpowiednie ustawienie parametrów symulatora klientów. Liczba klientów zostala ustalona na 240, odstep pomiedzy wywolaniami operacji zapisu na 15 sekund, a maksymalny czas opóznienia startu na 0. Charakterystyka uzyskanego strumienia jest przedstawiona na Rys. 42.



Rys. 42 Charakterystyka strumienia danych w konfiguracji (4*2*30,15,0).

W efekcie system byl periodycznie (co 15s) obciazany bardzo wysoka iloscia wywolan operacji zapisu. Uzyskane wyniki ilustruje Rys. 43.



Rys. 43 Sredni czas odpowiedzi modeli M2 i M3 w konfiguracji {as+bk=4,db=1}(4*2*30,15,0).

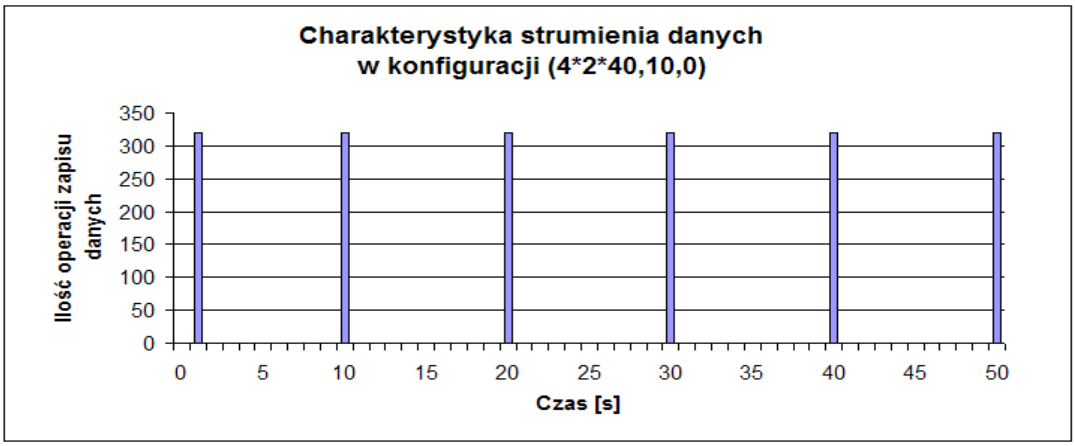
Na Rys. 43 pokazano srednie czasy odpowiedzi dla operacji rozpoczetych w kolejnych sekundach testu ($avgSRT_i$). Wyraznie zaznaczaja sie chwile czasu, w których wystepowalo duze obciazenie. Dla modelu M2 srednie czasy odpowiedzi sa zdecydowanie wieksze niz dla modelu M3. Jest to zachowanie zgodne z oczekiwanym, bowiem dzieki zastosowaniu brokera komunikatów w modelu M3 operacje zapisu danych do bazy danych sa wewnetrznie kolejkowane przez broker. Nalezy podkreslic, ze calkowite opróznienie kolejki nastepowalo kazdorazowo juz w 3-4 sekundzie po piku oraz ze we wszystkich przeprowadzonych testach stopa bledów (ERR) wynosila 0. Wykorzystana na Rys. 43 nazwa M3-NS, oznacza model M3 dzialajacy w oparciu o broker komunikatów zapisujacy komunikaty w pamieci operacyjnej. Charakterystyka tego modelu jest niemal identyczna z modelem M3, wykorzystujacym broker komunikatów zapisujacy komunikaty w lokalnym dla siebie pliku. Zachowanie takie swiadczy o wysokiej klasie stosowanego brokera oraz utwierdza w slusznosci przyjetego w punkcie 4.3.2 zalozenia odnosnie potrzeby utrwalania komunikatów na wypadek awarii.

Przeprowadzono dodatkowo eksperyment, w którym maksymalny czas opóznienia startu wynosil 15s, zapewniajac równomierny rozklad obciazenia. Test ten wykazal, ze ta sama ilosc danych, jednak podlegajaca innemu rozkladowi, moze byc zapisana przez oba systemy, z zachowaniem wymaganych parametrów jakosciowych, a wiec sredniego czasu odpowiedzi ponizej 500ms.

Przeprowadzone testy wykazuja, ze wykorzystanie brokera komunikatów powoduje zwiekszenie odpornosci systemu na chwilowy wzrost wielkosci strumienia danych. W wiekszej skali czasu nie poprawia ono jednak wydajnosci systemu. Ilosc danych, jakie mozna zapisac w systemie jest limitowana wydajnoscia bazy danych.

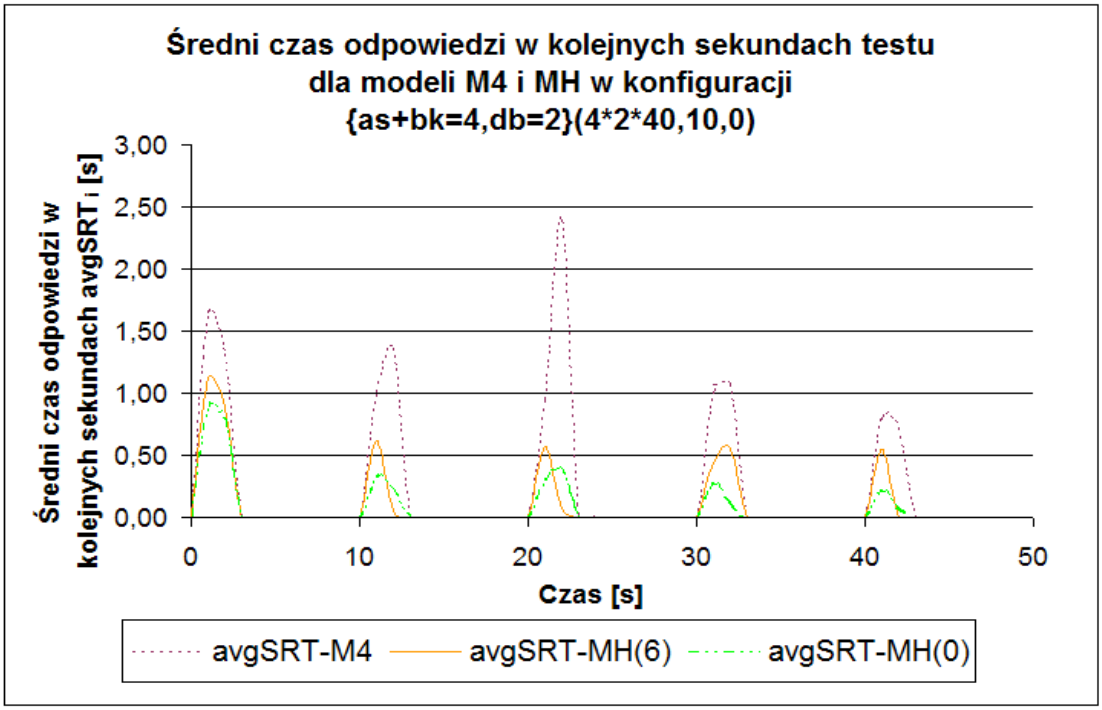
5.4.4 Wlasnosci modelu hybrydowego

Celem testu jest porównanie wlasnosci modelu M4 i MH poddawanych duzemu, chwilowemu obciazeniu. Model M4 partycjonuje dane i zapisuje je w sposób synchroniczny do kilku instancji baz danych. Model MH wykorzystuje dwa tryby zapisu: synchroniczny (tak jak w modelu M4) oraz asynchroniczny z wykorzystaniem brokera komunikatów. Oba modele wykorzystuja te sama konfiguracje {as+bk=4,db=2}. Zmienny strumien zostal wygenerowany, w podobny sposób jak w przypadku eksperymentu przedstawionego w punkcie 5.4.3. Uzyto konfiguracji (4*2*40,10,0), której charakterystyke przedstawia Rys. 47.



Rys. 44 Charakterystyka strumienia danych w konfiguracji (4*2*40,10,0).

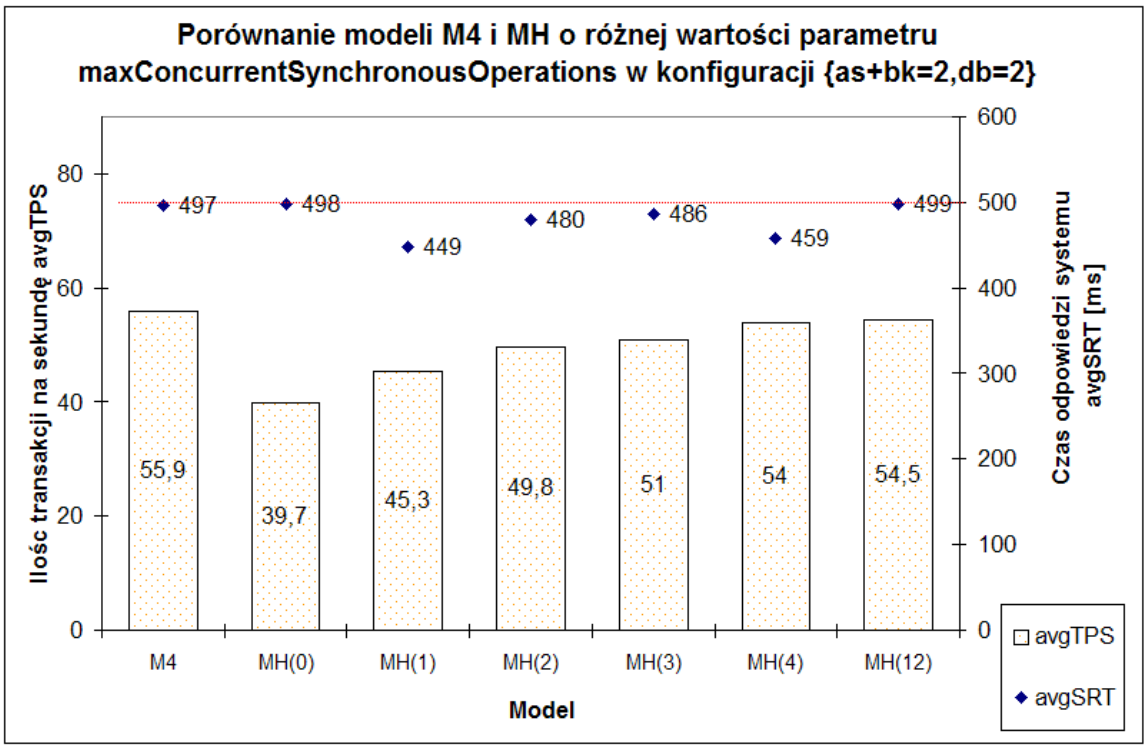
Uzyskane wyniki prezentuje Rys. 45.



Rys. 45 Sredni czas odpowiedzi dla modeli M4 i MH w konfiguracji {as+bk=4,db=2}(4*2*40,10,0).

Rys. 45 przedstawia srednie czasy odpowiedzi systemu ($avgSRT_i$) operacji rozpoczetych w kolejnych sekundach testu. Dodatkowe oznaczenie przy modelu MH jest wartoscia parametru *maxConcurrentSynchronousOperations,* czyli maksymalna, dopuszczalna iloscia równoczesnie wykonywanych operacji synchronicznego zapisu danych do bazy danych, której przekroczenie powoduje wykorzystanie trybu asynchronicznego, czyli przekazywanie danych do brokera komunikatów. Model MH(0) wykazuje duzo lepsza odpornosc na chwilowy wzrost obciazenia, uzyskane przez niego wartosci $avgSRT_i$ sa kilkukrotnie lepsze niz modelu M4. Taki wynik, model MH(0), zawdziecza ustawionej na 0 wartosci parametru

maxConcurrentSynchronousOperations, dzieki czemu wszystkie operacje byly wykonywane z uzyciem brokera. Pokazany na rysunku model MH(6), w których zwiekszono wartosc *maxConcurrentSynchronousOperations* do 6, wykazuje, zgodnie z oczekiwaniem, pogarszanie sie charakterystyki $avgSRT_i$. Zupelne pokrycie sie charakterystyk modelu MH i M4 nastepowalo przy wielkosci parametru równej 12. Wyniki testu moga sugerowac, ze dzialanie modelu hybrydowego wylacznie w trybie asynchronicznym (*maxConcurrent-SynchronousOperations* = 0) jest rozwiazaniem najkorzystniejszym. Niestety z wykorzystaniem trybu asynchronicznego wiaza sie dodatkowe narzuty, które powoduja pogorszenie wydajnosci systemu przy równomiernym obciazeniu. Przeprowadzono test porównujacy punkty pracy modelu M4 i MH, o róznej wartosci parametru *maxConcurrentSynchronous-Operations.* Uzyskane wyniki zawarte zostaly na Rys. 46.



Rys. 46 Porównanie modeli M4 i MH o róznej wartosci parametru maxConcurrentSynchronousOperations w konfiguracji {as+bk=2,db=2}.

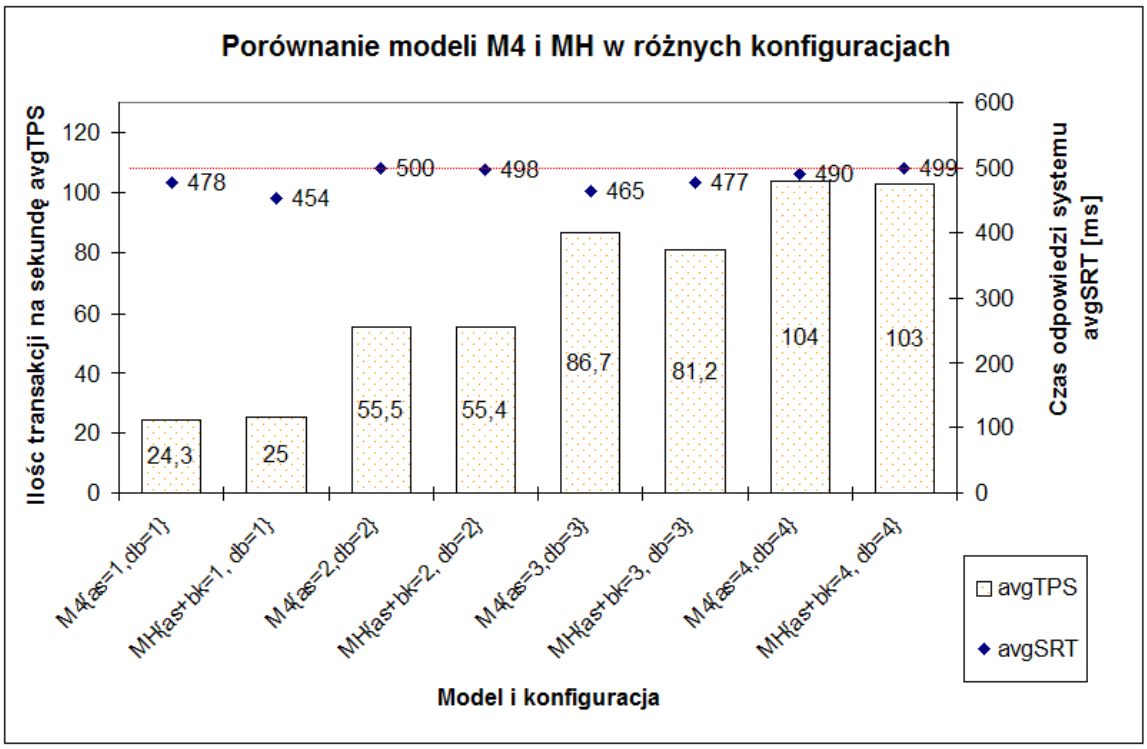
Zawarte na Rys. 46 dane obejmuja srednia ilosc transakcji na sekunde (avgTPS) oraz sredni czas odpowiedzi systemu (avgSRT) dla punktów pracy wyznaczonych dla modelu M4 i MH w kolejnych testowanych konfiguracjach poddawanych stalemu obciazeniu. Do wyznaczania punktu pracy modelu hybrydowego przyjeto dodatkowe zalozenie, ze dlugosc kolejki JMS musi sie w czasie testów utrzymywac na stalym poziomie. Uzyskane rezultaty pokazuja, ze z wykorzystaniem brokera komunikatów wiaze sie dodatkowe obciazenie serwera aplikacji,

które powoduje istotne (o okolo 30% dla modelu MH(0)) obnizenie ilosci przetwarzanych przez system w jednostce czasu transakcji. Zwiekszanie ‘udzialu’ operacji synchronicznych, poprzez zmiane wartosci parametru *maxConcurrentSynchronousOperations* powoduje, ze uzyskiwane przez model hybrydowy wartosci avgSRT zblizaja sie do uzyskanej przez model M4.

Przeprowadzone eksperymenty wykazuja przewage modelu hybrydowego nad modelem M4. Przyjeta w modelu hybrydowym koncepcja adaptowalnosci sposobu zapisu danych umozliwia dostosowanie charakterystyki dzialania modelu do aktualnego obciazenia. Dzieki temu model MH jest odporny na chwilowy wzrost wielkosci strumienia danych oraz posiada porównywalna z modelem M4 efektywnosc.

5.4.5 Skalowalnosc modeli M4 i MH

Celem testu jest zbadanie, na ile zastosowane w modelach M4 i MH(3) mechanizmy partycjonowania danych oraz wykorzystanie kilku instancji baz danych poprawiaja parametry wydajnosciowe oraz zapewniaja skalowalnosc. Wydajnosc obu modeli zostala porównana w nastepujacych konfiguracjach testowych: {as=1,db=1}, {as=2,db=2}, {as=3,db=3} oraz {as=4,db=4}. W kazdej z nich zostaly wyznaczone punkty pracy systemów. Uzyskane wyniki zaprezentowane sa na Rys. 47.



Rys. 47 Porównanie modeli M4 i MH w róznych konfiguracjach.

Rys. 47 przedstawia wartosci avgTPS oraz avgSRT dla poszczególnych modeli w kolejnych konfiguracjach. W wypadku obu modeli, wzrost krotnosci konfiguracji (rozszerzanie zasobów sprzetowych) powoduje wzrost sredniej ilosci transakcji (avgTPS) przetwarzanych przez model w ciagu sekundy. Wzrost ten ma charakter liniowy: dwukrotne zwiekszenie ilosci zasobów powoduje dwukrotny wzrost avgTPS. Jest to bardzo dobry wynik, oznacza on, ze **oba modele sa skalowalne**. Porównanie wartosci avgTPS pomiedzy konfiguracjami o tej samej krotnosci prowadzi do wniosku, ze oba modele maja zblizona wydajnosc, a zatem stosowane w modelu MH mechanizmy nie wnosza istotnych narzutów.

5.5 Podsumowanie

Jednym z najwazniejszych rezultatów jest opracowanie metodologii testów. Ze wzgledu na objetosc rozprawy nie przedstawiono srodowiska narzedziowego usprawniajacego wykonanie kilkuset testów. Opisano metodologie testów bazujaca na koncepcji punktu pracy systemu, sposób przeprowadzania testów, parametry uruchomieniowe oraz wymagane parametry jakosciowe. Uzyskane wyniki prowadza do nastepujacych wniosków:

- Koncepcja komponentu do grupowego zapisu danych jest wlasciwa. Wykazany w punkcie 5.4.1 wzrost wydajnosci modelu M1 uzyskany dzieki wprowadzeniu komponentu do zapisu grupowego jest bardzo znaczacy.
- Optymalizacja i strojenie systemu jest kluczowe. Uzyskany, w modelu M1 wzrost wydajnosci jest wiekszy niz uzyskiwany poprzez rozbudowywanie zasobów sprzetowych systemu. Optymalizacja i strojenie systemu pozostaja zatem dzialaniem podstawowym i kluczowym dla poprawy wydajnosci. Sa one mozliwe dzieki komponentowej budowie systemu, która pozwala na elastyczna, niezalezna od innych elementów systemu, wymiane dowolnego komponentu w juz istniejacym i dzialajacym systemie.
- Klasteryzacja serwera aplikacji, w przypadku aplikacji intensywnie zapisujacej dane nie powoduje podniesienia wydajnosci systemu. Zwiekszanie ilosci wezlów w klastrze serwerów aplikacji, na których uruchamiano model M2 nie przynioslo poprawy wydajnosci – punkt 5.4.2.
- Poprzez zastosowanie brokera komunikatów, mozna istotnie podniesc granice chwilowego obciazenia systemu, przy której zachowuje on zadane parametry

jakosciowe. Model M3, wykorzystujacy broker komunikatów, charakteryzuje sie duzo wyzsza odpornoscia na chwilowe zwiekszanie wielkosci strumienia danych niz model M2, który nie wykorzystuje brokera – punkt 5.4.3.

- W przypadku zastosowanego brokera komunikatów, nie obserwuje sie róznic w wydajnosci pomiedzy konfiguracja wykorzystujaca mechanizm utrwalania kolejki komunikatów w pliku i w pamieci operacyjnej.
- Sposób dzialania modelu hybrydowego mozna dopasowywac do charakterystyki obciazenia systemu. Zastosowane w modelu MH mechanizmy adaptacji sposobu zapisu danych umozliwiaja plynne balansowanie pomiedzy synchronicznym trybem zapisu, a trybem asynchronicznym wykorzystujacym broker komunikatów; pomiedzy mniejszymi opóznieniami zapisu, a wieksza odpornoscia na przeciazenia – punkt 5.4.4.
- Zastosowanie mechanizmów partycjonowania danych oraz wykorzystanie wielu instancji baz danych istotnie poprawia wydajnosc systemu. Wydajnosc modeli M4 i MH, wykorzystujacych te mechanizmy, rosnie wraz ze wzrostem krotnosci konfiguracji – punkt 5.4.5.
- Modele M4 i MH sa skalowalne. Uzyskiwana wydajnosc jest zalezna liniowo od krotnosci konfiguracji. Autor ma swiadomosc, ze dla kazdego z modeli nastapi kiedys odchylenie do wartosci stalej, której osiagniecie wyznaczy maksymalna ilosc zasobów, przy których nastepuje wzrost wydajnosci systemu. Wyznaczenie tej granicy wymaga jednak posiadania duzo wiekszych zasobów sprzetowych niz te, którymi dysponowal autor.

Najwazniejszym wynikiem przeprowadzonych badan eksperymentalnych, jest wykazanie skalowalnosci oraz okreslenie wydajnosci badanych modeli SZiPD. Wykazano, ze stworzony system SZiPD jest systemem skalowalnym i zapewnia odpowiednia dla tego typu systemów wydajnosc.

6 Zakonczenie i wnioski

Skonstruowany system zbierania i przechowywania danych pochodzacych z monitorowania systemów rozproszonych posiada wymagane dla tej klasy systemów wlasnosci co zostalo wykazane doswiadczalnie. System potrafi dzialac w warunkach duzej zmiennosci monitorowanych zasobów, które moga w dowolnym momencie zmieniac (dodawac, usuwac) parametry, jakie udostepniaja do monitorowania. Poprzez zaproponowanego agenta SZiPD, system moze laczyc sie z istniejacymi agentami systemów monitorujacych oraz zbierac dane w dowolnym z trybów: raportowanie, odpytywanie oraz sledzenie zmian wartosci; dane moga byc zapisywane z rózna czestotliwoscia. Posiadane wlasnosci funkcjonalne system zawdziecza, w glównej mierze, ogólnemu obiektowemu modelowi danych oraz uniwersalnym interfejsom dostepu. Spelnieniu wymagan odnosnie wydajnosci systemu poswiecone zostaly badania eksperymentalne. Pokazuja one, ze najbardziej dojrzaly sposród modeli – model hybrydowy, posiada odpowiednia wydajnosc, jest skalowalny oraz odporny na chwilowe zwiekszanie obciazenia.

Wykazano przydatnosci architektury komponentowej do tworzenia wydajnych i skalowalnych systemów zbierania i przechowywania duzej ilosc danych pochodzacych z monitorowania systemów rozproszonych. Optymalizacja systemów komponentowych w zakresie wydajnego zbierania i przechowywania danych nie zostala dotychczas szerzej przedstawiona w literaturze przedmiotu. Zebrane i usystematyzowane zostaly dostepne w tym obszarze rozwiazania, oraz wykorzystano je do konstrukcji kolejnych modeli systemu. Uzyskane wyniki moga byc interpretowane, nie tylko w kontekscie przedmiotowego systemu ale równiez, w sposób bardziej ogólny, jako porównanie mozliwosci w zakresie optymalizacji systemów komponentowych, intensywnie zapisujacych dane w bazie danych. Przyjeta metodyka testowania wydajnosci systemu komponentowego, oparta na definicji punktu pracy systemu, umozliwila wlasciwy pomiar oraz wykazanie najistotniejszych wlasnosci skonstruowanych modeli. Wyznaczanie punktów pracy jest podejsciem zgodnym z przyjeta definicja wydajnosci systemu i daje mozliwosc pokazania istotnych róznic pomiedzy modelami, przy zachowaniu narzuconych wymagan jakosciowych. Opracowanie i praktyczna weryfikacja uzytecznosci takiej metody do testowania wydajnosci systemów komponentowych jest waznym elementem proponowanego podejscia.

Systemy komponentowe sa bardzo dynamicznie rozwijajaca sie dziedzina informatyki. Juz w trakcie pracy nad rozprawa pojawily sie i zyskiwaly na popularnosci technologie EJB 3.0 oraz

Hibernate. Ich wykorzystanie do konstrukcji kolejnych modeli oraz porównanie wydajnosci wydaje sie bardzo ciekawe i gdyby autor dzis rozpoczynal prace z pewnoscia zostalyby uwzglednione. Nalezy jednak podkreslic, ze poruszane problemy architektury systemu oraz rozwiazania zwiazane z optymalizacja, pozostaja aktualne równiez w swietle tych dwóch nowych technologii. W pracy nie przedstawiono praktycznego wykorzystania systemu, które pozwoliloby na doswiadczalna weryfikacje jego uniwersalnosci i mozliwosci integracji z róznymi systemami monitorujacymi. Autor celowo pominal kwestie zwiazane z implementacja agenta SZiPD dla konkretnego systemu monitorujacego. W najblizszym czasie planuje bowiem implementacje rozwiazania ogólnego, które pozwoliloby na pobieranie danych z calej klasy systemów zgodnych ze standardem WBEM. Trwaja tez prace nad integracja SZiPD z systemem monitorujacym JIMS [ZJWB06]. Rysujace sie przed autorem kierunki badan i rozwoju systemu obejmuja wlasnie ten aspekt. Ciekawym wyzwaniem jest równiez automatyzacja stworzonych w modelu hybrydowym mechanizmów adaptowalnosci.

Sposób okreslania i zaspakajania wymagan aplikacji komponentowych, migracja komponentów, mechanizmy optymalnego rozmieszczania komponentów oraz przydzielania odpowiedniej liczby zasobów, podnoszenie wydajnosci systemu oraz uodparnianie na nierównomiernosc strumienia danych, bez potrzeby zwiekszania zasób sprzetowych, sa dzisiaj aktualnymi problemami stojacymi przed twórcami platform i aplikacji komponentowych. Ich rozwiazanie komplikuje ciagla presji na upraszczanie kwestii administracyjnych i konfiguracyjnych serwerów aplikacji oraz sposobów budowy samych komponentów. Aktualnie wykorzystywane sa jedynie proste mechanizmy, w sposób automatyczny dostosowujace wartosci wybranych parametrów konfiguracyjnych. Zdaniem autora przyszlosc nalezy do platform, które beda zaspakajac nie tyle statycznie zdefiniowane wymagania aplikacji odnosnie zasobów, ale okreslone wymagania jakosciowe. Platform, potrafiacych w sposób dynamiczny przydzielac zasoby na podstawie okreslonych polityk jakosci uslugi oraz mogacych ingerowac w konfiguracje i sposób dzialania poszczególnych komponentów. Mozliwosci adaptowania sposobu dzialania aplikacji komponentowej w celu dotrzymania parametrów jakosciowych, wykorzystane przez autora przy implementacji modelu hybrydowego, wpisuja sie w ten nurt oraz lokuja przeprowadzone prace w bardzo aktualnym kontekscie.

Zestawienie akronimów

API – ang. Application Programmer Interface
ASN1 – ang. Abstract Syntax Notation 1
BMP – ang. Bean Managed Persistence
BNF – ang. Backus Naur Form
CIM – ang. Common Information Model
CMP – ang. Container Managed Persistence
CORBA – ang. Common Object Request Broker Architecture
DTPS – ang. Data Throughput Per Second
EJB – ang. Enterprise Java Bean
EJB-QL – ang. Enterprise Java Bean Query Language
ERR – ang. error rate
HPC – ang. High Performance Computing
IDL – ang. Interface Definition Language
IP – ang. Internet Protocol
ISO – ang. International Organization for Standardization
IT – ang. Information Technology
ITU-T – ang. International Telecommunication Union - Telecommunication Standardization
JDBC – ang. Java DataBase Connectivity
JMS – ang. Java Messaging Service
JMX – ang. Java Management Extensions
JSR – ang. Java Specification Request
OOP – ang. Object Oriented Programming
ORB – ang. Object Request Broker
PDU – ang. Protocol Data Unit
POJO – ang. Plain Ordinary/Old Java Object
QoS – ang. Quality of Service
RDF – ang. Resource Description Framework
RFC – ang. Request For Comments
RFID – ang. Radio Frequency Identification
RMI – ang. Remote Method Invocation
RPC – ang. Remote Procedure Call
SNMP – ang. Simple Network Management Protocol
SOAP – ang. Simple Object Access Protocol
SQL – ang. Structured Query Language
SRT – ang. System Response Time
SSI – ang. Single System Image
SZiPD – System Zbierania i Przechowywania Danych
TCP – ang. Transmission Control Protocol
TPS – ang. Transaction Per Second
UDP – ang. User Datagram Protocol
UML – ang. Unified Modeling Language
URI – ang. Uniform Resource Indicator
URL – ang. Uniform Resource Locator
VPN – ang. Virtual Private Network

Bibliografia

Bibliografia zawiera 142 pozycje.

[AA99] Rahim Adatia, Faiz Arni Professional EJB, Wiley Computer Publishing, 1999

[AB03] Paul Allen, Joseph Bambara *Sun Certified Entrprise Architect for J2EE – study guide*, Osborne, 2003

[ABB04] Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock *The J2EE™ 1.4 Tutorial*, Sun Microsystems, 2004

[ACM01] Deepak Alur, John Crupi, Dan Malks *Core J2EE Patterns. Best Practices and Design Strategies*, Prentice Hall, 2001

[Ald04] Dave Aldridge, *A Practical Guide to Data Warehousing in Oracle*, DBAsupport.com, 2004, dostepny w Internecie: http://www.dbasupport.com// oracle/ora9i/datawarehousing01.shtml

[Amb04] Scott W. Ambler *Agile Database Techniques*, Wiley & Sons, 2004

[Ban98] Bela Ban *JavaGroups – group communication patterns in Java*, JavaGroups, 1998

[BEA04] Collective work *WebLogic Server 7.0 Administration Guide*, *Manging JDBC Connectivity*, BEA 2004, dostepny w Internecie: http://edocs.beasys.com/wls/docs70/ /adminguide/jdbc.html

[BK04] Christian Bauer, Gavin King, *Hibernate in Action*, Manning Publications Co., sierpien 2004

[BKP+00] Zoltán Balaton, Peter Kacsuk, Norbert Podhorszki, Ferenc Vajda *Comparison of Representative Grid Monitoring Tools*, white paper, 2000

[CC04] Collective work *Object Relational Tool Comparison* Cunningham & Cunningham, Inc., 2004

[CDB+00] R. G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, Fernando Velez *Object Data Standard – ODMG 3.0*, Morgan Kaufmann, 2000

[CDS+88] J.D. Case, M. Fedor, M.L. Schoffstall, J. Davin *Simple Network Management Protocol – RFC 1067*, 1988

[Cen04] Davor Cengija *Hibernate Your Data,* ONJava.com, 2004

[Cha04] John Chapman *Monitoring and Improving ASP.NET Application Performance* DnZone, 2004

[CL03] John Carnell, Jeff Linwood *Professional Struts Applications: Building Web Sites with Struts, Object Relational Bridge, Lucene, and Velocity*, Wrox Press, 2003

[CMP+05] Emmanuel Cecchet, Julie Marguerite, Mathieu Peltier, Nicolas Modrzyk *C-JDBC User's Guide*, 2005, dostepny w Internecie: http://c-jdbc.objectweb.org/ /current/doc/userGuide/html/index.html

[DB01a] David Deeths, Glenn Brunette *Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP*, Sun BluePrints, lipiec 2001

[DB01b] David Deeths, Glenn Brunette *Using NTP to Control and Synchronize System Clocks – Part II: Basic NTP Administration and Architecture*, Sun BluePrints, sierpien 2001

[DB01c] David Deeths, Glenn Brunette *Using NTP to Control and Synchronize System Clocks – Part III: NTP Monitoring and Troubleshooting*, wrzesien 2001

[Den02] Allen Denver, *Platform SDK documentation – PDH library*, Microsoft MSDN, 97, dostepny w Internecie: http://msdn.microsoft.com

[DHK+94] P. Dauphin, R. Hofmann, R. Klar, B. Mohr, A. Quick, M. Siegle, F. Sotz *ZM4/SIMPLE: a General Approach to Performance Measurement and Evaluation of Distributed Systems*, IEEE Computer Society Press, 1994

[DMTF00] Collective work *CIM Core White Paper*, DSP0111, 2000, dostepny w Internecie: http://www.dmtf.org/standards/published_documents.php

[DMTF06] Collective work *Web-Based Enterprise Management (WBEM)*, DMTF 2006, dostepny w Internecie: http://www.dmtf.org/standards/wbem/

[DMTF99] Collective work *Common Information Model (CIM) Specification, V2.2*, lipiec 1999, dostepny w Internecie: http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf

[Dor02] Paul Dorsey *Logical Partitioning of a Database: A New Way of Thinking about Enhancing Performance*, Dulcian, Inc., Maj 2002, dostepny w Internecie: http://www.dulcian.com/papers/Logical Partitioning of a Database.htm

[EC03] Expert Group Report *Next GenerationGrid(s) – European Grid Research 2005-2010*, EC document, 2003

[ESK+97] Greg Eisenhauer, Beth Schroeder, Karsten Schwan, Vernard Martin and Jeff Vetter, *DataExchange: High Performance Communication in Distributed Laboratories*, 9th International Conference on Parallel and Distributed Computing and Systems (PDCS'97), pazdziernik 1997.

[Eva04] Eric Evans *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Pearson Education Inc., 2004

[Focus03] Collective work *Application servers clustering - white paper*, Strategic Focus, grudzien 2003

[Fun00] Wlodzimierz Funika *Monitorowanie i analiza wskazników jakosci dzialania programów równoleglych na sieci stacji roboczych*, rozprawa doktorska, AGH 2000

[GHJ95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995

[Glo04] Jennifer Rodoni Glore *What's New in the J2EE Connector Architecture 1.5*, Sun Developer Network, 2004

[Gri05] Grinder Team, *The Ginder Home Page*, dostepna w Internecie: http://grinder.sourceforge.net

[GWB+04] Michael Gerndt, Roland Wismuller, Zoltan Balaton, Gabor Gombas, Peter Kacsuk, Zsolt Nemeth, Norbert Podhorszki, Hong-Linh Truong, Thomas Fahringer, Marian Bubak, Erwin Laure, Thomas Margalef *Performance Tools for the Grid: State of the Art and Future*, white paper, 2004

[Hae01] Richard Monson-Haefel, *Enterprise JavaBeans. Third Edition*, O'Reilly 2001

[HE98] Michael T. Heath, Jennifer A. Etheridge *Visualizing the performance of parallel programs*, 1998, dostepne w Internecie: http://www.cc.gatech.edu/computing/ /classes/cs7390_98_winter/reports/parallel/parag.html

[Heu03] Nick Heudecker *Introduction to Hibernate*, Systemmobile, grudzien 2003

[HM00] Steven L. Halter and Steven J. Munroe *Enterprise Java Performance*, Prentice Hall PTR, 2000

[HMR95] Michael T. Heath, Allen D. Mallony and Diane T. Rover *The Visual Display of Paraller Performance Data*, IEEE Computer, 1995

[Hor05] Jaroslaw Horodecki, *Wszystko pod kontrola*, Manager Magazine Grudzien 2005 Numer 12 (13), dostepny w Internecie: http://www.manager-magazin.pl/index.php?ct=archiwum&mag=13&art=11

[IBM04] Collective work *Performance Management Guide*, IBM, 2004, dostepny w Internecie: http://publib16.boulder.ibm.com/pseries/en_US/aixbman/prftungd/ multiprocess3.htm

[IBM04a] Collective work *Zarzadzanie infrastruktura IT*, IBM, 2004, dostepny w Internecie: http://www-5.ibm.com/shop/pl/software/tivoli/pdf/infrastruktura.pdf

[IBM04b] Collective work *IBM WebSphere Application Server, Advanced Edition Tuning Guide*, IBM 2004, dostepny w Internecie: http://www-306.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/was/0901.html#b198bb

[IBM05] Collective work *IBM Store Integration Framework Technology foundation for on demand retail solutions*, 2005, dostepny w Internecie: https://www.printers.ibm.com/sales/catalogs.nsf/vwImages/G581-0185-02/$file/ G581-018502.pdf?OpenElement

[Int04] Collective work, *Leveraging DEN-ng To Model All Business, System and Network Properties*, 2005, dostepny w Internecie: http://www.intelliden.com/ /page.asp?id=Insights&subID=DEN-ng

[Iona99] Collective work *Orbix 3 Programming Guide*, Iona Technologies Ltd., 1999

[Jas02] Mike Jasnowski *JMX Programming*, Wiley&Sons Inc, 2002

[JK04] Pawel Janik, Marcin Kielar *Automatyczna generacja warstwy instrumentacji w srodowisku JMX*, Praca Magisterska KI AGH, 2004

[JLV+03] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, Panos Vassiliadis *Hurtownie danych – podstawy organizacji i funkcjonowania*, WSiP 2003

[Jon03] Wiebe de Jong *Implement a JDBC Connection Pool via the Object Pool Pattern*, developer.com, dostepny w Internecie: http://www.developer.com/java/other/ /article.php/626291

[Jyt05] Jython Team, *The Jython Home Page*, dostepna w Internecie: http://www.jython.org/

[Kan01] Abraham Kang *J2EE clustering*, JavaWorld, wrzesien 2001

[KB02] Artur Kaszczyszyn Maciej Bajolek *Zastosowanie wzorców projektowych do tworzenia aplikacji w technologii J2EE*, praca magisterska, AGH 2002

[KB96] Kenneth P. Birman *Building Secure and Reliable Network Applications*, Manning 1996

[KG96] J.A. Kohl, G.A. Geist *The PVM Tracing Facility and XPVM*, Proceedings of the 29th Hawaii International Conference on System Science, Hawaii, USA 1996

[KS91] Carol E. Kilpatrick, Karsten Schwan *ChaosMON – Application-specific Monitoring and Display of Performance Information for Parallel and Distributed System*, Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, Santa Cruz, ACM Press, maj 1991

[Lam78] Leslie Lamport *Time, clocks and the ordering of events in distributed systems* Communications of the ACM, 1978

[Lar02] Craig Larman *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, Inc., 2002

[Lau01] Aleksander Laurentowski *A Distibuted Monitoring Service for Softvare Objects - rozprawa doktorska*, AGH, 2001

[Lau95] Aleksander Laurentowski *Patterns for a Unified Model of Heterogeneus Object-based Distributed Applications*, proceedings of European Research Seminar on Advances in Distributed Systems (ERSADS'95), 1995

[LB03] Sacha Labourey, Bill Burke *JBoss clustering*, grudzien 2003

[Li03] Sing Li *High-impact Web tier clustering, Part 1: Scaling Web services and applications with JavaGroups*, Wrox Press, 2003

[LRG00] Sai-Lai Lo, David Riddoch, Duncan Grisby *The omniORB version 3.0 User's Guide*, maj 2000

[LWS+03] T. Ludwig, R. Wismueller, V. Sunderam, A. Bode *OMIS – On-line Monitoring Interface Specification (v.2.1)*", Technische Universitaet Muenchen, Munich Germany, 2003

[Mar01] Floyd Marinescu *The State of The J2EE Application Server Marke*, The Server Side, 2001

[Mar02] Floyd Marinescu *EJB Design Patterns*, Wiley & Sons, 2002

[May04] Patric May *Evolutionary model driving integration*, AutevoMDI, 2004, dostepny w Internecie: http://www.intamission.com/docs/Autevo_Evolutionary_MDI.pdf

[MBW04] Thomas Mahler, Jakob Braeuchli, Armin Waibel *OJB - Basic Technique*, Apache DB project, 2004

[McC04a] Brian McCallister *Object Transaction Manager Tutorial*, Apache DB project, 2004

[McC04b] Brian McCallister *Persistence Broker Tutorial*, Apache DB project, 2004

[Mer03] Philippe Merle *OpenCCM: The Open CORBA Components Platform*, 3rd ObjectWeb Conference, INRIA Rocquencourt, France, listopad 2003

[MERQ04] Collective work *Monitoring J2EE Applications* analyst report, Mercury, lipiec 2004

[MK06] Dmitri Maximovich, Eugene Kuleshov, *High Performance Message Processing with BMT Message-Driven Beans and Spring*, Dev2Dev 2006, dostepny w Internecie: http://dev2dev.bea.com/pub/a/2006/01/custom-mdb-processing.html

[MM97] Thomas J. Mowbray, Raphael C. Malveau *CORBA Design Patterns*, Wiley & Sons 1997

[Mor02] Tadeusz Morzy *Przetwarzanie danych w magazynach danych*, V Seminarium PLOUG, Warszawa, 2002

[MR02] Celso L. Mendes, Daniel A. Reed *Monitoring Large Systems via Statistical Sampling*, Proceedings of the LACSI Symposium, Santa Fe, pazdziernik 2002

[MS03] Collective work *Windows 2000 Server Resource Kit - Praca serwera: Pomiary aktywnosci systemu wieloprocesorowego*, Microsoft, 2003, dostepny w Internecie: http://www.microsoft.com/poland/windows2000/win2000serv/PR_SER/default.mspx

[Mul02] Craig S. Mullins *Architectures for Clustering: Shared Nothing and Shared Disk*, DB2 Magazine, 2002, dostepny w Internecie: http://www.db2mag.com/db_area/archives/2002/q1/mullins.shtml

[MW04] Thomas Mathler, Armin Waibel *The ODMG Lock Manager*, Apache DB project, 2004

[Nau03] Brian Naughton *Deployment strategies focusing on massive scalability*, Sonic Software, 2003

[Nau60] Naur Peter, *Revised Report on the Algorithmic Language ALGOL 60*, *Communications of the ACM*, Vol. 3 No.5, pp. 299-314, May 1960.

[Noc04] Clifton Nock *Data Access Patterns: Database Interactions in Object-Oriented Applications*, Pearson Education Inc., 2004

[OF02] Collective work *Java Data Objects -Java Persistence Without the Pain*, OpenFusion, kwiecien 2002

[OH01] Piet Obermeyer, Jonathan Hawkins, *Microsoft dotNET Remoting: A Technical Overview*, MSDN - Microsoft Corporation, 2001

[OH97] Robert Orfali, Dan Harkey *Client/Server Programming with JAVA and CORBA* Wiley & Sons, 1997

[OMG04] Collective work *Common Object Request Broker Architecture: Core Specification*, OMG 2004, dostepny w Internecie: http://www.omg.org/docs/formal/04-03-12.pdf

[ORA05] Collective work *Oracle Database Advanced Replication10g Release 2*, Oracle, dostepny w Internecie: *http://download-uk.oracle.com/docs/cd/B19306_01/ /server.102/b14226/toc.htm*

[Plo99] Zdzislaw Plonski *Slownik Encyklopedyczny – Informatyka*, Wydawnictwa Europa, 1999

[PMJ01] B. Pop, R. Medesan, I. Jurca *Performance Comparison of Java Application Servers*, Tms, seria AC, 2001

[Pru05] Cameron Purdy *Architecting for Scalable Performances using Clustered Caching*, The Server Side Java Symposium, 2005, dostepny w Internecie: http://www.theserverside.com/symposium/presentations.html?News06_28_05-click#

[RAJ02] Ed Roman, Scott Ambler, Tyler Jewell *Mastering Enterprise JavaBeans. Second Edition*, Wiley & Sons, Inc., 2002

[Raj99] Gopalan Suresh Raj, *The CORBA Component Model (CCM)*, 1999, dostepny w Internecie: http://my.execpc.com/~gopalan/corba/ccm.html

[RC98] Michael Rosen, David Curtis *Integrating CORBA and COM Applications*, Wiley & Sons, 1998

[RDF04] Collective work *RDF Primer*, W3C 2004, dostepny w Internecie: http://www.w3.org/TR/ /2004/REC-rdf-primer-20040210/

[Ree01] Michael Reed *A Definition of Data Warehousing*, 2001 dostepny w Internecie: http://www.intranetjournal.com/features/datawarehousing.html

[Ric02] Jeffrey Richter *Microsoft .NET Framework Programminig*, Microsoft Press 2002

[RML03] Daniel A. Reed, Celso L. Mendes Charng-da Lu, Intelligent *Application Tuning and Adaptation - The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, Listopad 2003

[RR02] Dominik Radziszowski, Pawel Rzepa *Enterprise Java Beans*, TelenetForum, kwiecien 2002

[RSB05] Ed Roman, Rima Patel Sriganesh, Gerald Brose *Mastering Enterprise JavaBeans 3th edition*, Wiley&Sons Inc., 2005

[Sie01] Jon Siegel *Quick CORBA 3*, Wiley & Sons, 2001

[SKM+05] Srikanth Shenoy, Viswanath Krishnan, Nithin Mallya, Jagmohan Bhasin *J2EE Project Survival Guide*, Austin, 2005

[SL03] Sacha Labourey, Juha Lindfors, *JBoss Optimizations 101*, onJava.com, 2003, dostepny w Internecie: http://www.onjava.com/pub/a/onjava/2003/05/28/ /jboss_optimization.html

[Som05] Frank Sommers Upcoming Features in JDBC 4, Artima developper, 2005, dostepny w Internecie: http://www.artima.com/lejava/articles/jdbc_four.html

[Sonic04] Collective work *Benchmarking e-business messaging providers – white paper*, Sonic Software 2004

[SS05] Collective work *Application Server Matrix*, The Server Site, 2005

[SSJ+02] Inderjeet Singh, Beth Stearns, Mark Johnson, et al. *Designing Enterprise Applications with the J2EE Platform, Second Edition*, Sun Microsystems, 2002

[Sta93] William Stallings *SNMP, SNMP v2 and CMIP – The Practical Guide to Network-Management Standards*, Addison-Wesley, 1993

[Ste04] Ruth Stento *Persistent Data Developement Tools Validate the Model Driven Architecture Approach*, white paper, ObjectStore 2004

[Str03] John Strassner Policy-Based Network Management: Solutions for the Next Generation, Morgan Kaufmann, 2003

[Str03] Mark Strawmyer *.NET Remoting*, developer.com, 2003

[SUN02a] Collective work *Java 2 Platform, Enterprise Edition (J2EE) Overview*, SUN Microsystems 2002 dostepny w Internecie: http://java.sun.com/j2ee/ /setstandard.html

[SUN02b] Collective work *Java Transaction API (JTA) specification*, SUN Microsystems, 2002

[SUN02c] Collective work *JSR-77: Java 2 Platform, Enterprise Edition Management Specification*, SUN Micorystems, 2002

[SUN03a] Collective work *Java 2 Platform Enterprise Edition Specification, v1.2, 1.3, 1.4*, SUN Microsystems 2002, dostepny w Internecie: http://java.sun.com/j2ee/download.html#platformspec

[SUN03b] Collective work *JSR 160: JavaTM Management Extensions (JMX) Remote API 1.0 Specification*, Sun Microsystems, 2003

[SUN03c] Collective work *JSR 914: JavaTM Message Service (JMS) API*, SUN Microsystems, 2003

[SUN03d] Collective work *Sun ONE Application Server 7 Performance Tuning Guide*, SUN Microsystems, 2003, dostepny w Internecie: http://docs.sun.com/source/817-2180-10/index.html

[SUN04a] Collective work *Java Application Programmer Interface (API) 1.4* - Sun Microsystems 2004, dostepny w Internecie: http://java.sun.com/j2sdk/api/

[SUN04b] Collective work *JSR 12: JavaTM Data Objects (JDO) Specification*, Sun Microsystems, 2004

[SUN04c] Collective work *Enterprise JavaBeans Specification,version 2.1*, Sun Microsystems, 2004

[SUN04d] Collective work *JSR003: Java Management Extension (JMX) Maintenance Release 2*, Sun Microsystems, 2004, dostepny w Internecie: http://jcp.org/aboutJava/communityprocess/mrel/jsr003/index2.html

[SUN04d] Collective work *SUN N1 System – documentation*, Sun Microsystems, 2004, dostepny w Internecie: http://www.sun.com/software/n1gridsystem/docs.xml,

[SUN04e] Collective work *SUN Microsystems N1™ Grid Service Provisioning System*, white paper, dostepna w Internecie: http://www.sun.com/software/whitepapers/service_provisioning/ApplicationAwarenessWP.pdf

[SUN05] Collective work *JSR170: Content Repository API*, SUN Microsystems, 2005, dostepny w Internecie: http://wiki.aparzev.com/confluence/download//attachments//2021/jsr170-1.pdf?version=1

[SW03] Benjamin G. Sullins, Mark B. Whipple *EJB Cookbook*, Manning 2003

[SYS95] S. R. Sarukkai, J. C. Yan, M. Schmidt *Automated Instrumentation and Monitoring of Data Movement in Parallel Program*, Proceedings of the 9th International Parallel Processing Symposium, Santa Barbara, CA., kwiecien 1995

[TSL+00] J. Trinitis, V Sunderam, T. Ludwig, R. Wismueller *Interoperability Support in Distributed On-line monitoring Systems*, Proceedings of 8th European Conference on High Performance Computing and Networking (HPCN EUROPE 2000), Amsterdam, Springer, maj 2000

[WAC+03] Joseph Walnes, Ara Abrahamian, Mike Cannon-Brookes, Patrick A. Lightbody *Java Open Source Programming: with XDoclet, JUnit, WebWork, Hibernate*, Wiley, 2003

[WB03] Robert Wrembel, Bartosz Bebel *Oracle - Projektowanie rozproszonych baz danych*, Helion, wrzesien 2003

[Wir82] Wirth Niklaus, *Programming in Modula-2*, Berlin, Heidelberg: Springer, 1982.

[Wir89] Niklaus Wirth *Algorytmy + struktury danych = programy*, WNT, 1989

[WL96] R. Wismueller, T. Ludwig *TOOLSET – An integrated Tool Environment for PVM*, Proceedings of HPCN'96, Brussels, Springer, 1996

[Woz02] Brian Wozny *EJB Checksum insted of Version Number Pattern?*, theServerSide, 2002, dostepny w Internecie: http://www.theserverside.com/patterns/ /thread.tss?thread_id=13199

[WR03] Craig Walls, Norman Richards *XDoclet in Action*, Manning Publications Co, grudzien 2003

[Wyb05] Gazeta Wyborcza *Rozmowy przechowywane,* Agora, 2005-07-14

[Yog00] Yogesh Malhotra *Knowledge Management and Virtual Organizations* Idea Group Publishing, 2000

[Yu05] Wang Yu *Uncover the hood of J2EE clustering*, theServerSide, sierpien 2005, dostepny w Internecie: http://www.theserverside.com/articles/ /article.tss?l=J2EEClustering

[ZAO02] Peter Zadrozny, Philip Aston, Ted Osborne, *J2EE Performance Testing*, APress, 2003

[ZJWB06] Krzysztof Zielinski, Marcin Jarzab, Damian Wieczorek, Kazimierz Balos *JIMS Extensions for Resource Monitoring and Management of Solaris 10*, International Conference on Computational Science (4) 2006: 1039-1046

[ZL02] Krzysztof Zielinski, Aleksander Laurentowski *Integracaja Systemów B2B: JMX*, TelenetForum, kwiecien 2002

[ZLS+95] Krzysztof Zielinski, Aleksander Laurentowski, Jakub Szymaszek, Andrzej Uszok *A tool for monitoring heterogenous distributed object applications*, proceedings of 15th International Conference on Distributed Computing Systems, Vancouver, IEEE CS Press. Maj 1995