STUDIA INFORMATICA 2005
Volume 26 Number 1 (62)

Marcin GORAWSKI, Jan PALICA
Politechnika Śląska, Instytut Informatyki

IMPLEMENTACJA PRZESTRZENNEJ TELEMETRYCZNEJ HURTOWNI DANYCH W TECHNOLOGII ORACLE/JINI

Streszczenie. W pracy przedstawiono system Przestrzennej Telemetrycznej Hurtowni Danych wykorzystujący RDBMS Oracle oraz technologię JINI. Do systemu wprowadzono nowy model komunikacji oparty na przepływie obiektów, zastępując poprzedni model oparty na RMI. Zaprezentowano podstawy teoretyczne technologii JINI, zalety i wady nowego rozwiązania oraz porównano wydajność obu systemów.
Słowa kluczowe: hurtownia danych, JINI, JavaSpaces

IMPLEMENTATION OF SPATIAL TELEMETRIC DATA WAREHOUSE USING ORACLE/JINI TECHNOLOGY

Summary. In this paper we present Spatial Telemetric Data Warehouse which uses RDBMS Oracle and JINI technology. We replace previous way of communication based on RMI standard and introduce a new way of communication which is modeled as flow of object. We describe the basics of JINI technology, show advantages and disadvantages of a new solution and compare both models.
Keywords: data warehouse, JINI, JavaSpaces

1. Wprowadzenie

System przestrzennej telemetrycznej hurtowni danych (ang. Spatial Telemetric Data Warehouse (STDW)) zbiera dane telemetryczne o pomiarach zużycia mediów (gazu, prądu, wody, ciepła) [1, 2, 3].

Rozwiązanie stanowi dwuwarstwowa infrastruktura telemetryczna, którą tworzy:

a) telemetryczny system zintegrowanego odczytu liczników,
b) system przestrzennych hurtowni danych telemetrycznych.

Telemetryczny system zintegrowanego odczytu liczników bazuje na technologii AMR (ang. Automated Meter Reading) oraz GSM/GPRS. System ten pozwala przesyłać dane z liczników zużycia mediów zlokalizowanych na dużym obszarze geograficznym do serwera telemetrycznego z wykorzystaniem sieci telefonii komórkowej GSM w technologii GPRS. Podstawowe zalety bezprzewodowych systemów GSM/GPRS w przypadku zdalnego rozliczania zużycia mediów to niskie koszty i krótki czas wdrożenia, dowolna odległość pomiędzy licznikami, niewrażliwość na ukształtowanie terenu, brak rozbudowanych systemów antenowych oraz możliwość natychmiastowego powiadamiania o sytuacjach awaryjnych.

System przestrzennych hurtowni danych telemetrycznych (STDW) jest systemem wspomagania podejmowania decyzji taktycznych dotyczących wielkości produkcji medium na podstawie krótkoterminowych prognoz ich zużycia. Prognozy te sporządzane są na podstawie analiz danych zgromadzonych w hurtowni danych zasilanej z serwera telemetrycznego. Serwer telemetryczny jest źródłem danych pomiarowych, które w procesie ekstrakcji są ładowane do bazy STDW.

System STDW opisany jest modelem gwiazdy kaskadowej [16] indeksowanej aR-drzewem na platformie Oracle9i/Java [2].

System STDW(RMI) w swojej podstawowej architekturze z jednym klientem i kilkoma serwerami działa w wyniku wywoływania metod zdalnych RMI (ang. Remote Method Invocation). Klient, przesyłając zadanie do serwerów, tworzy dla każdego z nich osobny wątek. Wątki są odpowiedzialne za utrzymanie komunikacji z poszczególnymi serwerami. Klient zna dokładne adresy wszystkich serwerów i porty, na których nasłuchują rejestry RMI.

W niniejszym artykule przedstawiono system STDW w rozproszonej architekturze o modelu komunikacji opartej na technologii JavaSpaces i JINI 2.0 (ozn. STDW(JINI)). Przy pewnych założeniach upraszczających budowę aR-drzewa prezentowany model komunikacji w systemie STDW(JINI) może być dostosowany do pracy wielu klientów i wielu serwerów.

Systemy STDW(RMI) i STDW(JINI) posiadają taką samą funkcjonalność, a ich architektura jest transparentna dla użytkownika końcowego.

Implementacja systemu STDW(JINI) oparta na technologii JINI miała na celu pokonanie ograniczeń, jakie niesie ze sobą standardowe RMI. Zmiana sposobu komunikacji z modelu RMI na model oparty na przepływie obiektów przez przestrzeń JavaSpaces miała na celu sprawdzenie funkcjonalności nowej metody komunikacji oraz porównanie wydajności obu metod. Praca ta jest pierwszym etapem przeniesienia funkcjonalności JINI do systemu hurtowni danych.

1.1. Wywoływanie zdalnych metod

RMI jest metodą komunikacji aplikacji zaimplementowanych w Javie, działających na różnych wirtualnych maszynach. W systemach rozproszonych każdy węzeł może wykonywać różne zadania zdalnie wywoływane. Zdalne wykonywanie zadań w uproszczeniu przedstawia się następująco:

- wysłanie przez klienta żądania wykonania procedury (wraz z podaniem odpowiednich argumentów) do serwera aplikacji,
- wykonanie procedury przez serwer,
- przesłanie przez serwer wyniku działania procedury do klienta,
- wykorzystanie przez klienta otrzymanego wyniku działania zdalnej procedury do dalszych zadań.

Wcześniejszą technologią wywoływania zdalnych funkcji było RPC (ang. Remote Procedure Call). W RPC zastosowano podejście strukturalne (niezależne procedury) z jednoczesną realizacją żądań niezależną od języka programowania i typu platformy. RPC wymagała np. konwersji typów danych (różne rozmiary liczb całkowitych) lub konwersji reprezentacji typów danych (różny zapis liczb całkowitych – ang. little endian, big endian). Poważną wadą RPC był brak możliwości przesyłania złożonych typów danych.

RMI zaprojektowane zostało do pracy w homogenicznym środowisku aplikacji Javy. Ponieważ RMI stosuje podejście obiektowe, zmieniono w nazwie technologii słowo procedura (procedure) na metoda (method), wskazując w ten sposób na możliwość zdalnego wykonywania zadań przypisanych dla obiektu. RMI stanowi dobre rozwiązanie w przypadku zdalnego wykonywania zadań realizowanych w Javie pomiędzy platformami różnego typu.

1.2. JINI i JavaSpaces

JINI jest nową technologią sieciową [4]. W architekturze JINI serwery (usługodawcy) rejestrują swoje usługi w serwisie typu lookup, do którego klienci wysyłają zapytania w poszukiwaniu pożądanych usług.

Jedną z możliwości przeniesienia funkcjonalności technologii JINI do opisywanego systemu STDW jest stworzenie serwisu zarządzającego serwerami. Serwis zarządzający jest odpowiednim rozwiązaniem, ponieważ serwery udostępniają swoje usługi w grupie, a nie każdy oddzielnie. Klient nie wyszukuje serwerów osobno, ale chce uzyskać dostęp do grupy serwerów udostępniających daną usługę. Serwis taki zarządza grupą serwerów i rejestruje w serwisie lookup usługę, która umożliwiałaby wykorzystanie grupy serwerów. Przyglądając się bliżej samej komunikacji systemu, można dojść do wniosku, że klienci z serwerami wymieniają się obiektami. JINI wspiera rozwiązania oparte na przepływie obiektów (ang. flow of object), udostępniając implementację technologii JavaSpaces. Rozwiązanie to

wprowadza do systemu pośrednika (serwis JavaSpaces) między klienta i serwery. Od tej pory nie istnieje bezpośrednie połączenie między klientem a serwerami. Wszystkie operacje wykonywane są za pośrednictwem serwisu JavaSpaces. Wprowadzając taki rodzaj komunikacji uniezależniono klienta od wiedzy na temat serwerów.

Technologia JINI 2.0 opiera się na zdalnych interfejsach, obiektach proxy i wywoływaniach zdalnych metod, a komunikacja z serwisem JavaSpaces oparta jest na nowym standardzie zdalnego wywoływania metod zwanym JERI (ang. Jini Extensible Remote Invocation). Zastosowanie JINI 2.0 daje możliwość pokonania ograniczeń standardu RMI.

2. Technologia JINI

JINI jest zbiorem klas, interfejsów, narzędzi programistycznych i pewnych konwencji tworzenia aplikacji działających w środowisku rozproszonym, które wspomagają tworzenie i rozmieszczanie systemów sieciowych. Na architekturę tej technologii składają się trzy komponenty: model programowania, środowisko wykonawcze (ang. runtime) infrastruktury i serwisy [4].

Odpowiednie API i konwencje technologii JINI, które wspierają model zdalnych zdarzeń, rozproszonych transakcji oraz dzierżawy, pozwalają na tworzenie solidnych i pewnie działających systemów rozproszonych, mimo niestabilności i zawodności sieci.

- Model programowania – jest zbiorem interfejsów klas umożliwiających tworzenie solidnych i niezawodnych serwisów, jest zbiorem konwencji tworzących pracującą infrastrukturę. Stanowi rozszerzenie standardowego modelu programowania m. in. o następujące interfejsy:
  - Interfejs dzierżawy (ang. leasing interface) – określa sposób zajmowania i zwalniania zasobów.
  - Interfejs zdarzeń i zawiadamiania (ang. event and notification) – który jest rozszerzeniem modelu zdarzeń znanego z komponentów JavaBeans przeniesionym do środowiska rozproszonego.
  - Interfejs transakcji – dostarcza narzędzi do nadzorowania grup operacji; albo wykonają się wszystkie operacje w grupie albo żadna.
- Runtime infrastruktury (ang. runtime infrastructure) – składa się protokołów sieciowych, m.in. tzw. „odnajdź i przyłącz” (ang. discovery and join) – oraz implementujących je API, umożliwia w łatwy sposób dodawanie, lokalizacje, dostęp i usuwanie urządzeń oraz serwisów z całego systemu rozproszonego tworzącego infrastrukturę JINI. Obejmuje rozproszone mechanizmy bezpieczeństwa, które są zintegrowane z RMI – jest to rozszerzenie standardowych mechanizmów

bezpieczeństwa platformy Javy dla systemów rozproszonych. W skład tej infrastruktury wchodzi serwis typu „lookup”, stanowiący repozytorium wszystkich serwisów tworzących całą infrastrukturę JINI.

- Serwisy – są głównym elementem infrastruktury JINI. Serwisem może być dowolne urządzenie umieszczone w sieci, które spełnia określone zadania i funkcje. Może nim być dowolna aplikacja, która dostarcza określonych usług. Najczęściej są to aplikacje napisane w Javie, komunikujące się poprzez RMI. Serwis implementuje interfejsy, które definiują zadania i usługi realizowane w ramach danego serwisu.

Technologia JINI może być postrzegana jako sieciowe rozszerzenie infrastruktury, modelu programowania i narzędzi dostarczanych wraz ze standardową technologią Javy. W tabeli 1 przedstawiono porównanie odpowiadających sobie elementów:

Tabela 1

Architektura technologii JINI [1][4]

	Infrastructure	Programming Model	Services
Base Java	Java VM	Java APIs	JNDI
	RMI	JavaBeans	Enterprise Beans
	Java Security	...	JTS
Java + Jini	Discovery/Join	Leasing	Printing
	Distributed Security	Transactions	Transaction Manager
	Lookup	Events	JavaSpaces Service

Dostarczany przez firmę Sun Microsystems, Inc. pakiet startowy JINI Technology Starter Kit v2.0_002 zawiera większość elementów wymaganych do stworzenia infrastruktury JINI. W pakiecie znajdują się specyfikacje wszystkich komponentów i środków programowych. Pakiet zawiera API umożliwiające tworzenie serwisów i komponentów korzystających ze wszystkich dostępnych środków programowych. Implementacje środków programowych dostarczane są w postaci pakietów jar. W celu stworzenia infrastruktury JINI można skorzystać ze standardowych serwisów, których podstawowa implementacja jest dostarczana z pakietem startowym. Wraz z pakietem dostępne są również wszystkie kody źródłowe.

2.1. Środowisko wykonawcze infrastruktury

Środowisko wykonawcze infrastruktury jest umiejscowione w trzech podstawowych modułach rozproszonej infrastruktury JINI, tj.:

- Serwis – dostawca usług, np. drukarka, repozytorium danych, serwer obliczeniowy.
- Klient – element, który korzysta z usług dostarczanych przez serwisy.
- Serwis typu lookup – pośrednik między serwisami a klientami, centralne repozytorium informacji o zarejestrowanych serwisach.

Wszystkie te elementy są połączone ze sobą poprzez sieć za pośrednictwem protokołu TCP/IP. JINI jest teoretycznie niezależna od protokołów, ale na razie jedyna implementacja jest oparta na TCP/IP. Kod jest transmitowany pomiędzy tymi elementami poprzez tzw. szeregowanie obiektów. (ang. marshalling). Aby obiekty mogły poruszać się w sieci, muszą być serializowane. Wysyłanie i odbieranie obiektów jest wspierane przez tzw. gniazda (ang. socket).

RMI dostarcza mechanizmu, który umożliwia wyszukanie, uruchomienie i usunięcie zdalnych obiektów. Pozwala także, aby pomiędzy obiektami w sieci przekazywane były obiekty razem z kodem, a nie tylko same dane. Większość udogodnień technologii JINI jest osiągana dzięki możliwości przesyłania kodu poprzez sieć.

Mechanizm bezpieczeństwa w technologii JINI opiera się na pojęciach użytkownika (ang. principal) i listy kontroli dostępu. Dostęp do serwisu jest uzależniony od zawartości listy kontroli dostępu do obiektu. Serwis może żądać dostępu do innego serwisu bazując na tożsamości, jaką ten obiekt implementuje [4].

Dostęp do usług wielu serwisów w infrastrukturze JINI jest oparty na mechanizmach dzierżawy. Dzierżawa jest gwarantowanym czasem dostępu do danej usługi. Okres dzierżawy jest negocjowany pomiędzy klientem a serwisem, zanim będzie można skorzystać z danych usług. Przydzielony okres dzierżawy nie musi być równy pożądanemu. Jeżeli dzierżawa nie zostanie odnowiona z jakiegokolwiek powodu lub żądanie przedłużenia nie zostanie przyjęte, to zasoby zostają zwolnione. Dzierżawa może być wyłączna lub współdzielona. Przy dzierżawie wyłącznej na czas zajmowania zasobu nie jest on dostępny dla innych klientów. W przypadku dzierżawy współdzielonej dostęp do zasobu jest nieograniczony.

Grupa operacji w obrębie jednego serwisu lub kilku może być wykonana jako jedna transakcja. Serwis transakcji implementuje interfejsy koordynujące transakcje dwufazowe (ang. two phase commit).

Obiekt może pozwolić innemu obiektowi zarejestrować słuchacza, który powiadomi zdalny obiekt o wystąpieniu zdarzenia.

2.2. Rejestracja serwisów i proces „discovery and join”

Serwis (usługa) jest w pewnym sensie pojęciem logicznym. Zwykle serwis jest definiowany lub identyfikowany poprzez interfejsy, które implementuje. Każdy serwis może być zaimplementowany na wiele sposobów lub przez wielu dostawców. Tym, co łączy serwisy lub sprawia, że są identyczne, jest taki sam interfejs, który implementują. Serwisy mogą się różnić tym, że każda implementacja może używać innego zbioru obiektów lub obiektów należących do różnych klas.

Przez samo pojęcie serwis rozumiemy obiekt, który dostarcza pewnych usług. To czy serwis będzie eksportował obiekt tzw. proxy (obiekt za pomocą, którego wykonywana jest

komunikacja ze zdalnym serwisem), czy będzie eksportował samego siebie, zależy od implementacji. Pojęcie serwis i dostawca usług można stosować zamiennie, ponieważ zadania, jakie wykonuje dostawca usług, są częścią całości ogólnie rozumianej jako serwis. Dostawca usług ma do wykonania następujące zadania:

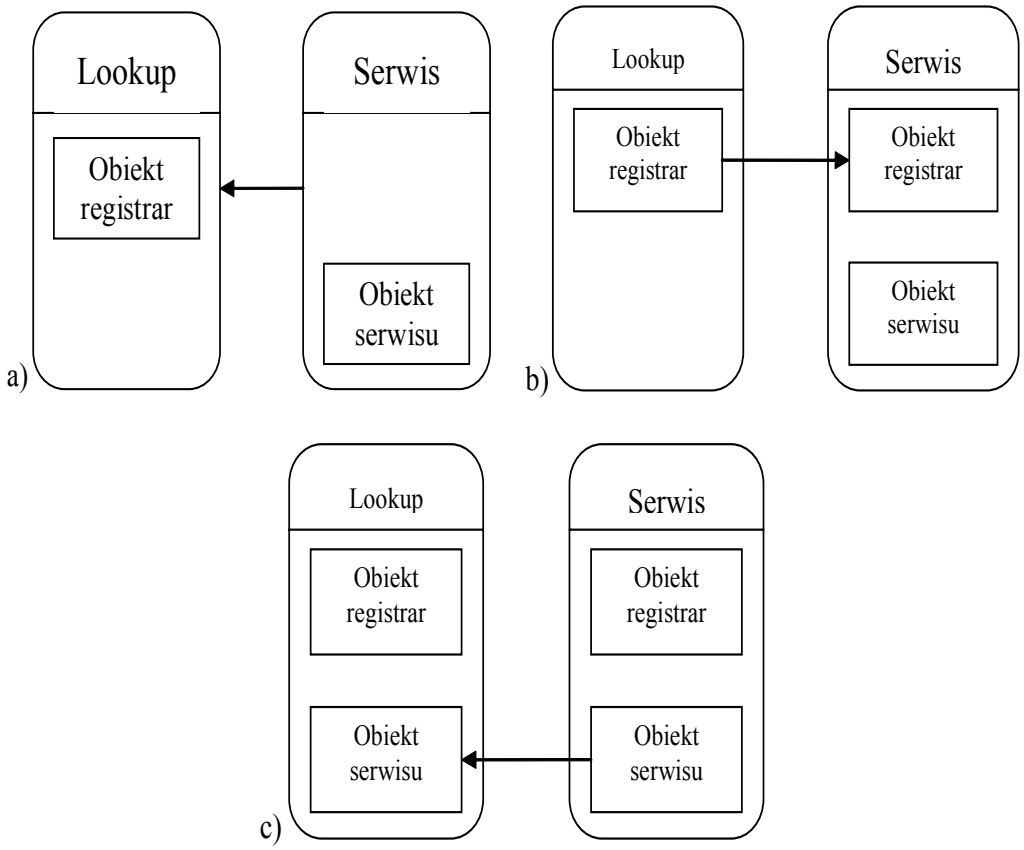
- Stworzyć serwis - usługę; stworzy obiekty implementujące serwis.
- Zarejestrować samego siebie i usługi w serwisie typu lookup.
- Pozostać aktywnym, aby umożliwić stały dostęp do usług.

Obiekt serwisu, który jest rejestrowany w serwisie typu lookup, jest nazywany obiektem serwisu, zdalnym obiektem serwisu lub obiektem proxy.

Aby serwis mógł zarejestrować swój obiekt zdalny w serwisie typu lookup, musi go najpierw zlokalizować. Standardowy port, na którym nasłuchuje serwis typu lookup, to 4160. Lokalizacja tego serwisu może odbywać się na dwa sposoby: w trybie bezpośrednim lub trybie rozsyłania grupowego. Tryb bezpośredni używany jest wtedy, kiedy znany jest adres IP lub URL i serwis może ustanowić bezpośrednie połączenie TCP do serwisu typu lookup. Wyszukiwanie w trybie grupowym odbywa się poprzez wysłanie pakietów UDP przez serwis. Nasłuchujący na porcie 4160 serwis typu lookup odpowie na żądanie wysyłając swój obiekt zdalny. Obiekt proxy serwisu typu lookup, zwany „registrar”, implementuje interfejs „ServiceRegistrar”. Jakiekolwiek żądania kierowane do serwisu typu lookup wykonywane są za pośrednictwem metod tego interfejsu.

Od momentu, kiedy serwis jest w posiadaniu obiektu, registrar serwisu typu lookup może zarejestrować swoje usługi i stać się członkiem infrastruktury JINI. Aby dołączyć do infrastruktury, serwis musi wywołać metodę register() obiektu proxy serwisu typu lookup, przekazując jako parametr swój obiekt zdalny, tzw. ServiceItem oraz obiekty opisujące serwis. Metoda register() wysyła kopię obiektu proxy serwisu w postaci obiektu ServiceItem, który jest składowany w serwisie typu lookup. Po tych czynnościach proces dołączania jest zakończony i usługi serwisu są zarejestrowane w serwisie typu lookup.

Obiekt klasy ServiceItem jest kontenerem obiektów, tzn. oprócz samego obiektu zdalnego serwisu, za pośrednictwem którego klienci komunikują się z serwisem, zawiera jeszcze inne obiekty, które są atrybutami serwisu – obiekty opisujące serwis, ułatwiające jego wyszukiwanie. Proces „discovery and join” jest przedstawiony na rysunku 1.



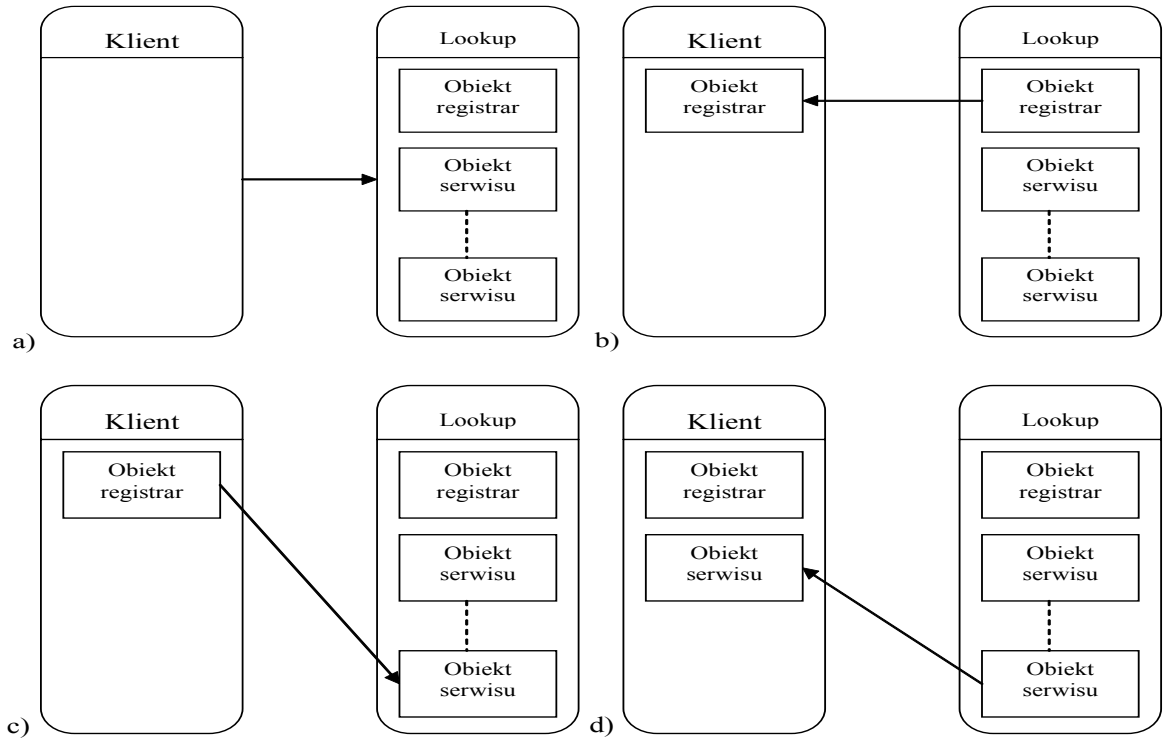
Rys. 1. Proces „discovery and join" [4,15]: a) wyszukanie serwisu typu lookup, b) pobranie obiektu registrar, c) zarejestrowanie obiektu zdalnego serwisu

Fig. 1. Discovery and join: a) querying for a service locator, b) registrar returned, c) service uploaded

2.3. Wyszukiwanie (lookup process)

Od momentu zarejestrowania usługi (serwisu) w serwisie typu lookup jest ona dostępna dla klientów. W rozproszonym środowisku serwisów klient musi mieć możliwość zlokalizowania właściwego serwisu i pobrania jego obiektu zdalnego. Proces zlokalizowania i pobrania obiektu „registrar" z serwisu typu lookup jest taki sam jak opisany wcześniej. Klient, posiadający już obiekt registrar, aby zacząć wyszukiwanie serwisów musi skorzystać z metody lookup() interfejsu ServiceRegistrar, który ten obiekt implementuje. Parametrem wywołania tej metody jest obiekt klasy ServiceTemplate. Jest to obiekt, którego zadaniem jest przekazanie serwisowi typu lookup kryteriów, według których ma zostać wyszukany pożądany serwis. Aby znaleźć serwis, który implementuje interfejsy przez nas wymagane, należy stworzyć tablicę obiektów typu Class, przyporządkowując jej odpowiednie interfejsy i przekazać ją jako parametr do klasy ServiceTemplate. Kolejnym atrybutem klasy ServiceTemplate jest ServiceId. Jest to obiekt jednoznacznie identyfikujący usługę

w serwisie typu lookup. Dodatkowo serwis może być opisany za pomocą obiektów implementujących interfejs Entry. Przekazanie tablicy obiektów tego typu do obiektu ServiceTemplate umożliwia zawężanie kryteriów wyszukiwania. Wywołanie funkcji lookup() powoduje wysłanie obiektu ServiceTemplate i w przypadku powodzenia wyszukania zwracana jest kopia obiektu proxy szukanego serwisu. W przypadku niepowodzenia zwracana jest wartość null. Proces wyszukiwania jest przedstawiony na rysunku 2.



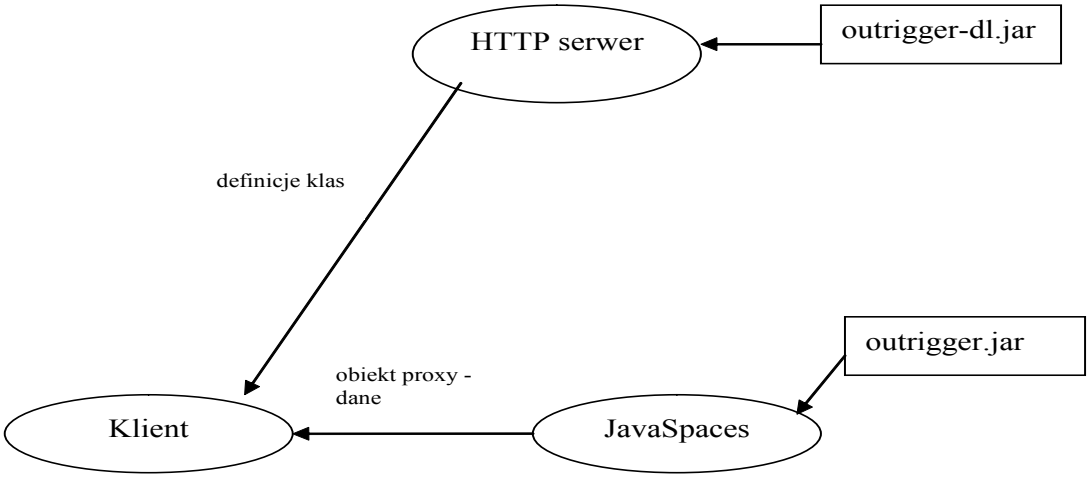
Rys. 2. Wyszukanie i pobranie obiektu zdalnego serwisu [4,15]:
a) wyszukanie serwisu typu lookup, b) pobranie obiektu registrar, c) przekazanie kryteriów wyszukiwania, d) pobranie obiektu zdalnego serwisu

Fig. 2. Asking and getting proxy object:
a) querying for a service locator, b) registrar returned
c) asking for a service, d) service returned

2.4. Serwer HTTP

Zdalny obiekt proxy serwisów jest w pierwszej kolejności eksportowany do serwisów typu lookup, a następnie pobierany przez klientów. Obiekt proxy jest przekazywany do serwisu typu lookup w formie serializowanej. Taka kopia obiektu proxy jest przywracana do życia (deserializowana) dopiero na wirtualnej maszynie javy (JVM) klienta. Obiekt składa się z kodu i danych (nie może być zrekonstruowany z samych danych). Dlatego do deserializacji

wymagany jest również kod. Gdyby kod był po stronie klienta, to cała rozproszona infrastruktura JINI straciłaby swą elastyczność. Niemożliwe byłoby wtedy dodanie nowego urządzenia do infrastruktury, ponieważ aktualni klienci nie potrafiliby zdeserializować obiektu proxy. Aby uniknąć takiej sytuacji w infrastrukturze JINI definicje klas wymaganych do deserializowania są ściągane z serwerów, zwykle z tego samego miejsca, z którego pochodzi serwis. Istnieje kilka sposobów ściągania definicji klas, a najbardziej popularne jest wykorzystanie protokołu HTTP lub FTP. Serwis, do którego należy obiekt proxy, ma zdefiniowany tzw. „codebase", który posiada również obiekt zdalny serwisu. Codebase zawiera informacje o lokalizacji definicji klas i dzięki temu klient jest w stanie ściągnąć odpowiednie biblioteki i zdeserializować obiekt. Jeżeli codebase zawiera adres URL, wtedy musi być dostępny pod tym adresem serwer HTTP, zawierający wymagane definicje klas. Nie jest wymagane, aby dla każdego serwisu pracował osobny serwer HTTP. Na jednym takim serwerze mogą znajdować się biblioteki różnych serwisów. Dla przykładu serwis JavaSpaces dostarczany przez firmę SUN składa się z dwóch pakietów jar (rys. 3.)



Rys. 3. Pobieranie obiektu proxy i definicji klas [15]
Fig. 3. Support services for JavaSpaces

3. Specyfikacja JavaSpaces

Przestrzeń JavaSpaces operuje na tzw. wpisach. Wpis jest pewnego rodzaju grupą obiektów, w języku Java wyrażony jako klasa, która implementuje interfejs net.jini. core.entry.Entry. Wpis w postaci kopii może być zapisany do przestrzeni JavaSpaces, którą zarządza serwis JavaSpaces. Wyszukanie wpisu w serwisie JavaSpaces polega na stworzeniu obiektu szablonu (ang. template), który również jest wpisem. Szablon jest wpisem, którego nie wszystkie pola muszą być określone. Przy wyszukiwaniu wpisu w przestrzeni JavaSpaces

biorą udział tylko te pola szablonu, które mają określoną wartość – wartości tych pól muszą odpowiadać dokładnie wartościom pól wyszukiwanego wpisu. Istnieją dwa rodzaje operacji wyszukania wpisu w przestrzeni JavaSpaces: czytaj (ang. read) i pobierz (ang. take). Wywołanie funkcji czytaj powoduje zwrócenie wpisu, który dokładnie pasuje do szablonu, lub informacji o braku takiego wpisu w przestrzeni. Działanie funkcji pobierz jest takie same jak funkcji czytaj, jednak dodatkowo wpis jest usuwany z przestrzeni JavaSpaces.

Zarządzający przestrzenią serwis JavaSpaces umożliwia powiadamianie zarejestrowanych klientów o interesujących ich właśnie zapisanych do przestrzeni wpisów. Możliwość ta istnieje dzięki wykorzystaniu modelu rozproszonych zdalnych zdarzeń zaimplementowanych przez serwis JavaSpaces.

Architektura technologii JavaSpaces wspomaga mechanizmy transakcji, pozwalające na wykonywanie niepodzielnie wielu operacji na przestrzeni JavaSpaces. Model „two phase commit" transakcji jest zaimplementowany w pakiecie net.jini.core.transaction. Dodatkowo ważnym elementem jest czas życia wpisu w przestrzeni JavaSpaces, który jest określony przez dzierżawy.

Architektura systemu rozproszonego oparta na implementacji technologii JavaSpaces odpowiada modelowi przepływu obiektów, ponieważ obiekty są transmitowane do i z przestrzeni JavaSpaces.

Szereg operacji czytania, pobierania, zapisywania, powiadomienia jest wykonywanych przez klientów na serwisach implementujących technologię JavaSpaces. Operacje mogą być pojedyncze lub zebrane w grupy. Grupy operacji mogą być wykonywane jako jedna transakcja, tzn. albo wykonają się wszystkie operacje, albo żadna. Pojedynczy klient może pracować jednocześnie z dowolną ilością serwisów JavaSpaces. Klient może rejestrować się jako słuchacz w oczekiwaniu na interesujący go wpis. Powiadomienia są przekazywane do słuchaczy zdarzeń, którymi mogą być nie tylko sami klienci, ale również obiekty proxy.

3.1. JavaSpaces i bazy danych

Przestrzeń JavaSpaces nie jest ani relacyjną ani obiektową bazą danych. JavaSpaces ma pomagać w rozwiązywaniu problemów w rozproszonych środowiskach obliczeniowych, a nie służyć jako repozytorium danych. Relacyjna baza danych przechowuje dane i operuje na nich bezpośrednio za pomocą języka zapytań. JavaSpaces przechowuje wpisy, posiada wiedzę o typie tych wpisów i o zserializowanej formie każdego z pól. Nie istnieje żaden ogólny język zapytań. Pole albo pasuje do pola szablonu, albo nie. Dodatkowo JavaSpaces pracuje tylko na kopiach wpisów. JavaSpaces jest zaprojektowana jako prosta składnica obiektów. Funkcjonalność JavaSpaces mieści się gdzieś pomiędzy systemem plików i bazami danych.

Technologia JavaSpaces zapewnia:

- platformę wspomagającą projektowanie i implementacje rozproszonych systemów obliczeniowych,
- nieskomplikowany model klienta – wymagane klasy i interfejsy pobierane z zewnątrz,
- możliwość tworzenia różnorodnych implementacji,
- możliwość tworzenia repliki serwisu JavaSpaces,
- działanie tych samych wpisów i szablonów niezależnie od implementacji.

3.2. Operacje interfejsu JavaSpaces

Wszystkie operacje wykonywane są na obiekcie implementującym interfejs net.jini.space.JavaSpace. Każda implementacja serwisu JavaSpaces eksportuje obiekt proxy, który ten interfejs implementuje. Wszystkie operacje wykonywane przez klienta są wykonywane poprzez obiekt proxy. Obiekt ten znajduje się po stronie klienta.

Istnieją cztery rodzaje operacji, jakie mogą być wykonywane przez serwis JavaSpaces. Każda operacja posiada jako jeden z parametrów wpis lub szablon. W tym podrozdziale szczegółowo zostaną opisane następujące operacje:

- Write – zapisuje dany wpis do serwisu JavaSpaces.
- Read – pobiera kopię wpisu z serwisu JavaSpaces pasującego do danego szablonu.
- Take – pobiera i usuwa wpis z serwisu JavaSpaces pasujący do danego szablonu.
- Notify – powiadamia zarejestrowany obiekt, że wpis pasujący do zarejestrowanego szablonu został zapisany do serwisu JavaSpaces.

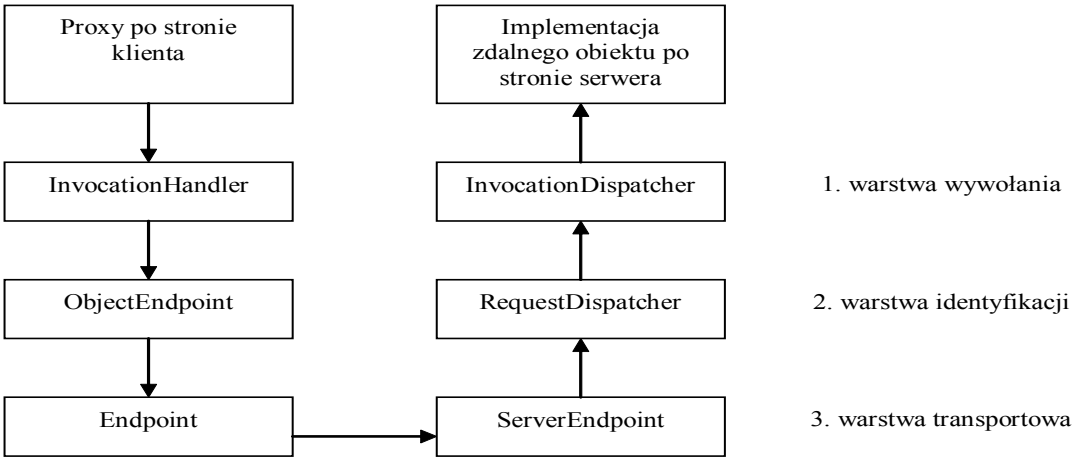
Termin „operacja” odnosi się do pojedynczego wywołania metody serwisu JavaSpaces.

3.3. Jini Extensible Remote Invocation – JERI

Model JERI łączy wspólna semantyka RMI i cechują go:

- nowy model bezpieczeństwa,
- wyeliminowanie generowania plików stub przez zewnętrzny kompilator rmic,
- możliwość wykorzystania innych protokołów niż TCP,
- poprawione rozproszone zbieranie nieużytków (garbage collection),
- możliwość dostosowywania stosu protokołów do własnych potrzeb.

Celem JERI jest odsłonięcie stosu protokołów, aby umożliwić wpływ na prawie każdy aspekt wywoływania metody RMI. Aby wykorzystać te możliwości, należy zrozumieć sposób, w jaki wywoływana metoda przechodzi przez poszczególne warstwy stosu protokołów. Na rysunku 4 są przedstawione poszczególne warstwy.



Rys. 4. Stos protokołów RMI [7]
Fig. 4. The RMI protocol stack

Mechanizm eksportowania jest odpowiedzialny za stworzenie obiektu proxy, który po stronie klienta jest odpowiedzialny za poszczególne kroki przechodzenia przez kolejne 3 warstwy stosu.

Pierwszą warstwą inicjującą wywołanie zdalnej metody jest warstwa zwana Invocation layer lub Marshaling layer. Kiedy klient wywoła zdalną metodę, obiekt proxy w tej pierwszej warstwie przekazuje wywołanie metody do obiektu InvocationHandler, który jest odpowiedzialny za tzw. szeregowanie argumentów (marshalling – przekształcenie do odpowiedniej postaci, umożliwiające przekazanie do zdalnej maszyny) lub rozszeregowanie zwracanych wartości. Kiedy wywołanie jest odebrane przez serwer, obiekt Invocation-Dispatcher jest odpowiedzialny za pobranie zdalnego obiektu, rozszeregowanie, wywołanie metody z pobranymi parametrami i w końcu szeregowanie zwracanych wartości tak, że wyniki mogą zostać wysłane do klienta.

Warstwą środkową jest bardzo cienka warstwa identyfikacji i służy ona do identyfikacji zdalnego obiektu, z którego ma zostać wywołana zdalna metoda. Jest ona również odpowiedzialna za rozproszone zbieranie nieużytków.

Ostatnią warstwą jest warstwa transportowa, która jest odpowiedzialna za zakodowanie zdalnego wywołania metody w odpowiednim formacie i wysłanie jej przez sieć do serwera. Warstwa transportowa jest zaimplementowana dla różnego rodzaju protokołów komunikacji: TcpEndpoint, HttpEndpoint, SslEndpoint. Warstwa transportowa nie jest świadoma warstw powyższych i dane, które posiada, są dla niej tylko zbiorem bajtów. Jest po prostu odpowiedzialna za transport bajtów.

JERI pozwala na modyfikowanie pierwszej i trzeciej warstwy. Warstwa druga jest na tyle cienka, że nie jest warta wprowadzania zmian. W warstwie pierwszej istnieje możliwość wprowadzania dużych zmian. Możemy przekazywać dodatkowe parametry, różnorako

implementować metody, argumenty, dodać kontrolę dostępu. Podsumowując, JERI umożliwia zmianę tylko części warstw lub pozwala zastępować je własnymi. Można również tworzyć podklasy, które jednocześnie będą korzystały z klas InvoctionHandler i InvocationDispatcher.

Następnym udogodnieniem dla programisty jest automatyczna generacja obiektów proxy. Nie ma potrzeby używać kompilatora rmic do generowania plików stub.

Bezpieczeństwo w JINI jest osiągnięte przez ingerowanie w stos protokołów za pomocą tzw. secure JERI. Jest właściwie identyczne z JERI i często korzysta z tych samych klas. Dzięki możliwości ingerencji w kod do niektórych klas dodano nowe metody, w niektórych metodach dodano nowe argumenty. Obecnie jest zaimplementowana warstwa transportowa, która wykorzystuje SSL/TLS i Kerberos.

JINI udostępnia elastyczny mechanizm konfiguracji, który pozwala np. zdefiniować eksporter, protokoły transmisji i inne opcje aplikacji w czasie wdrażania.

JINI 2.0 jest nową infrastrukturą sieciową, która wprowadza do technologii zdalnych wywołań metod RMI rozwiązania, których wcześniej brakowało. Technologia JINI nie tylko poprawiła niedociągnięcia starego RMI, ale również wprowadziła nowe rozwiązanie w postaci dynamicznej konfiguracji.

4. System rozproszony STDW(JINI)

System STDW(JINI) składa się z trzech zasadniczych modułów, którymi są aplikacja klientów, aplikacja serwerów w połączeniu z hurtownią danych opartą na RDBMS Oracle 9i oraz serwisy technologii JINI w wersji 2.0. Moduł klienta jest przeznaczony do zarządzania pracą systemu i przeprowadzania interakcji z użytkownikiem, podczas gdy moduł serwera może wykonywać większość algorytmów obliczeniowych, pozwalając na rozłożenie obciążenia na wszystkie serwery i odciążając w ten sposób komputery, na których będą pracowały moduły klienta. Wykorzystanie technologii JavaSpaces umożliwiło w prosty sposób rozszerzenie systemu STDW o dostęp dla wielu klientów. Rozproszona struktura systemu DSDW(JINI) opiera się na standardzie klient-serwer.

4.1. Klient

Zadaniem modułu klienta jest przedstawienie użytkownikowi informacji dotyczących zużycia mediów, zarządzanie zadaniami serwerów i składanie poszczególnych wyników agregacji. Interfejs użytkownika zaimplementowany w systemie bazuje na zestawach map regionów, w których są zlokalizowane liczniki i z których mają być pobierane pomiary. Użytkownik systemu za pomocą „gumowego prostokąta” definiuje podregion (tzw. okno

agregacji), z którego mają zostać zebrane agregaty. W systemie STDW jest zaimplementowany specjalny algorytm, który odpowiednio dzieli nachodzące na siebie okna. Proces agregacji rozpoczyna się w momencie przesłania listy okien agregacji do serwerów. W momencie uruchamiania modułu klienta następuje wyszukanie serwisu lookup i pobranie zdalnych obiektów serwisu JavaSpaces i transakcji. Serwisy te biorą czynny udział w procesie komunikacji między serwerami a klientami. Klient wie tylko, gdzie jest serwis JINI. W serwisie JavaSpaces umieszcza wpisy z oknami agregacji. Następnie klient czeka na wyniki. O nadejściu poszczególnych wyników informuje go serwis JavaSpaces. Jeśli serwis JavaSpaces wyśle powiadomienie do klienta, to klient pobiera wpis z przestrzeni. Klient czeka tak długo, aż pobierze wszystkie wyniki. Klient wie, ile jest serwerów w systemie rozproszonym, co pozwala mu stwierdzić, czy pobrał wszystkie wyniki. Po pobraniu wyników z poszczególnych serwerów klient składa wyniki i pokazuje je użytkownikowi w formie stabelaryzowanej.

4.2. Serwer

Moduł serwera STDW jest umieszczony na każdym z komputerów stanowiących system rozproszony. Zadaniem serwera jest komunikacja z bazą danych, składowanie danych, generowanie list liczników oraz budowanie drzew agregatów. Dzięki obiektowi zarządzania drzewem agregatów klient może stworzyć na serwerze takie drzewa, jakich parametry ustawi w swoim pliku konfiguracyjnym. Może również, podając zbiór okien agregacji, otrzymać wartości zagregowanych danych zgromadzonych w drzewie na serwerze. Obsługa modułu serwera ogranicza się tylko do uruchomienia go i podania parametrów niezbędnych do pracy (użytkownik, hasło i instancja bazy danych, adres serwera lookup, ustawienie numeru porządkowego serwera). Część tych informacji podaje się w plikach konfiguracyjnych serwera lub na formatce w momencie uruchamiania. O aktualnym stanie modułu serwera informuje wyświetlana formatka, na której są zgromadzone wszystkie interesujące użytkownika informacje – region obejmowany przez serwer, liczba wykonanych akcji agregacji itd.

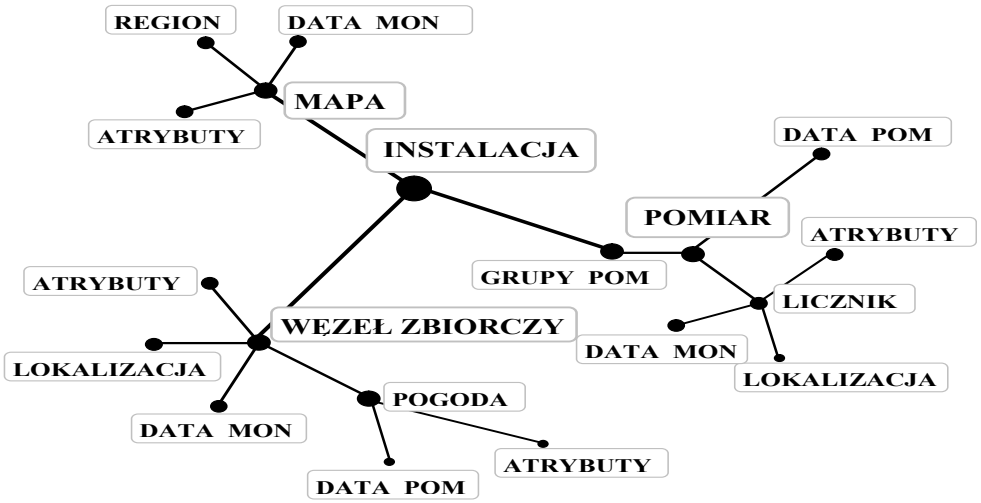
4.3. Serwisy JINI

Serwisy JINI oraz serwer WWW są trzecim istotnym elementem systemu STDW(JINI). W systemie tym są wykorzystywane trzy standardowe serwisy JINI: serwis lookup, transakcji i JavaSpaces. Komunikacja między klientem a serwerem jest teraz oparta na przepływie obiektów, a rolę pośrednika pełni serwis JavaSpaces.. Wszystkie zadania, okna agregacji i żądania przesyłane są do serwera w postaci obiektów typu wpis poprzez serwis JavaSpaces. Również serwery, wszystkie wyniki i agregaty przesyłają bezpośrednio do serwisu

JavaSpaces. Dla klientów i serwerów nie jest istotne to, gdzie się wzajemnie znajdują. Klienci i serwery w momencie startu muszą znać adres serwisu lookup, który jest pewnym centrum informacji o wszystkich dostępnych serwisach w infrastrukturze JINI. Serwis transakcji w tym systemie odgrywa rolę nadzorcy i dba o to, aby pewne operacje, w których udział biorą klienci, serwis JavaSpaces i serwery wykonały się w całości. Jego zadaniem jest również kierowanie ruchem, tzn. pilnowanie, żeby wyniki wysłane do jednego klienta nie zostały przechwycone przez innego. Serwer WWW jest elementem wymaganym przez technologię JINI i jego rola opisana jest w rozdziale dotyczącym technologii JINI. Serwisy JINI muszą zostać uruchomione, zanim zostaną uruchomione moduły serwerów i klientów.

4.4. Projekt bazy danych

Przeznaczeniem systemu STDW jest pobieranie i analiza dużej liczby przestrzennych danych telemetrycznych zamodelowanych w schemacie gwiazdy kaskadowej [16] przedstawionym na rys. 5.



Rys. 5. Schemat gwiazdy kaskadowej użytej w systemie [2]
Fig. 5. Cascade star diagram [2]

Schemat gwiazdy kaskadowej użytej w systemie składa się z głównej tabeli faktów Instalacja, która łączy ze sobą trzy główne wymiary MAPA, POMIAR oraz WĘZEŁ ZBIORCZY. Wymiar MAPY składuje informacje dotyczące map regionów, w których zlokalizowane są liczniki. O mapach są przechowywane następujące informacje: rodzaj (mapa wektorowa lub bitmapowa), charakterystyka obejmowanego regionu (geograficzna lokalizacja, rozmiary), data stworzenia mapy, która pozwala na wymianę nieaktualnych map. Wymiar WĘZEŁ ZBIORCZY przechowuje dokładne informacje o lokalizacji i parametrach danego węzła. Podwymiar POGODA zawiera informacje dotyczące pogody, takie jak:

wilgotność, temperatura, zachmurzenie, których rolą w późniejszych analizach jest zbadanie wpływu zużycia mediów w zależności od pogody.

Największą liczbą danych obciążony jest wymiar POMIAR. Pojedynczy odczyt z licznika wiąże się z wysłaniem do WĘZŁA ZBIORCZEGO następujących danych: data i czas odczytu, identyfikator licznika i odczytana wartość. Odczytana wartość jest uzależniona od rodzaju danego licznika, poza tym dla liczników energii elektrycznej i wody istnieją dwie strefy czasowe, dla liczników gazu jedna. Ponieważ każdy WĘZEŁ ZBIORCZY zarządza pewną grupą różnego rodzaju liczników, tabela GRUPA POMIARÓW łączy główną tabelę faktów z wymiarem POMIAR. Dzięki temu możliwa jest łatwa identyfikacja pomiaru i licznika w zależności od węzła zbiorczego, do którego należy licznik. Tabela LICZNIK jest tabelą, która zawiera informację o liczniku, tzn. dacie montażu, lokalizacji i atrybutach.

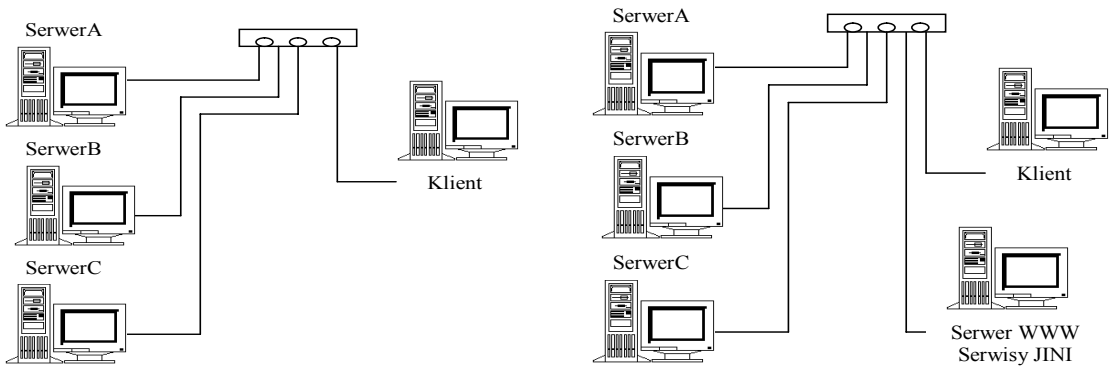
Gwiazda kaskadowa jest indeksowana aR-drzewo (drzewo agregatów). Opis algorytmu konstrukcji aR-drzewa oraz algorytmów bezpośrednio z nim związanych znajduje się w [2].

Oprócz użycia nietypowych indeksów, jakimi są drzewa agregatów tworzone przez moduł serwera, tabele w bazie danych są indeksowane za pomocą standardowych technik, jakie dostarcza system Oracle 9i.

5. Testy wydajnościowe

Szczegółowe porównanie wydajności systemów STDW(RMI) i STDW(JINI) wymagało stworzenia odpowiedniej jednolitej struktury węzłów i połączeń dla obu systemów.

Architekturę systemu STDW(RMI) i STDW(JINI) obrazują odpowiednio rysunki 6 .



Rys. 6. Struktura systemu opartego na RMI (z lewej), technologię JINI (z prawej)
Fig. 6. RMI based structure (left) JINI based structure (right)

W obydwu wersjach systemów trzy serwery pracowały jako rozproszona hurtownia danych. W systemie STDW(JINI) dodatkowo pracował serwer, na którym były uruchomione serwisy JavaSpaces, lookup, transakcji i dodatkowo prosty serwer WWW. Komputer klienta

był taki sam w obydwu wersjach. Komputery pracowały w sieci lokalnej ethernet o prędkości transmisji 100Mb/s. Zaznaczyć również należy, że komputery nie były specjalnie przygotowywane, tzn. nie były usuwane z nich inne procesy. Konfiguracja sprzętowa i programowa komputerów była następująca:

- Klient – Pentium 4 2,8GHz, 512MB RAM; Windows XP, Java Sun 1.4.2_04, aplikacja klienta.
- Serwer A – Pentium 4 2,8GHz, 512MB RAM; Windows XP, Java Sun 1.4.2_04, Oracle 9i, aplikacja serwera.
- Serwer B i C – Pentium 4 1,7GHz, 256MB RAM; Windows 2000, Java Sun 1.4.2_04, Oracle 9i, aplikacja serwera.
- Serwer serwisów JINI™ – Pentium 4 1,7GHz, 256MB RAM; Windows 2000, Java 1.4.2_04, serwisy Reggie, Mahalo, Outrigger, serwer Http dla plików *.jar i *.class.

5.1. Założenia

Testowanie miało na celu uchwycenie różnic czasowych w komunikacji systemu STDW opartego na serwisie JavaSpaces i RMI. Dla wszystkich testów została przyjęta jednakowa wysokość drzewa agregacji, natomiast zmieniane były okresy agregacji [2]. Wymiary regionu wynosiły 2000 X 2000. Na każdym z serwerów znajdowały się dane z dwóch węzłów zbiorczych, co miało zapewnić w miarę jednakową liczbę danych na każdym serwerze. Dodatkowo okna agregacji były tak dobrane, aby do mocniejszego Serwera A było przydzielonych kilka liczników więcej w porównaniu z serwerami słabszymi. Okna agregacji obejmowały losowo wybrane liczniki. Aplikacja klienta zaczynała mierzyć czas w momencie startu procesu agreguj – moment wysłania okien agregacji do serwerów, a kończyła w chwili odebrania i połączenia wszystkich agregatów przez klienta.

5.2. Zasady testowania

Testy podzielono na dwie grupy. Testy pierwszej grupy polegały na pobieraniu danych, które wcześniej nie znajdowały się w drzewie agregacji, tzn. wszystkie dane musiały być pobrane z bazy danych. Pozostałe testy pobierały te same agregaty, z tym że dane te wcześniej były pobrane do pamięci serwera. Podejście takie miało na celu obserwacje wpływu natychmiastowej reakcji serwerów na komunikację. Dla obu systemów testy obejmowały odpowiednio okres agregacji 1/2/3 miesiące – pomiary były wykonywane dla 10, 20, 30 i 40 liczników.

Liczba liczników, jaka przypadała na poszczególny serwer, była uzależniona od całkowitej liczby liczników w pomiarze (tabela 2).

Tabela 2

Liczba liczników przypadająca na poszczególny serwer w zależności od całkowitej liczby liczników w pomiarze

	10	20	30	40
Serwer A	6	8	12	16
Serwer B	4	6	9	12
Serwer C	4	6	9	12

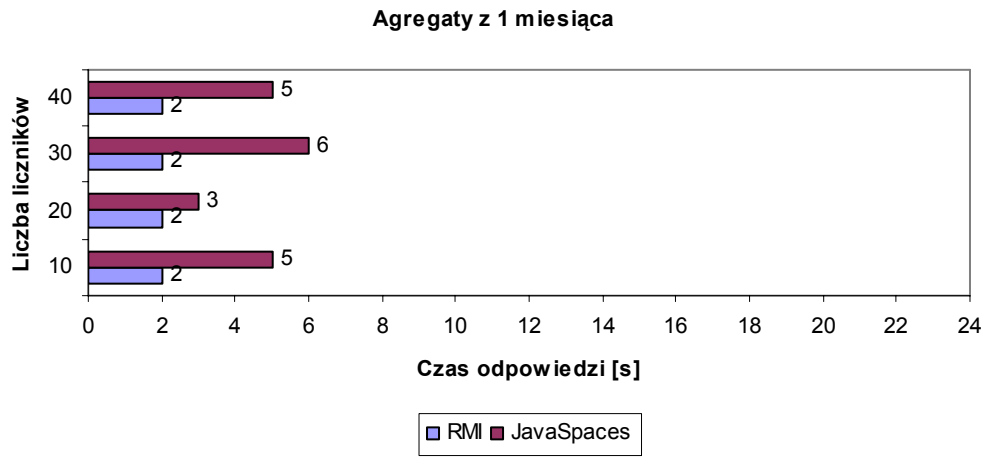
5.3. Analiza wyników

Przed przystąpieniem do testów wiadomo było, że komunikacja oparta na JavaSpaces musi być trochę wolniejsza od RMI, ponieważ serwery nie komunikują się za pośrednictwem serwisu JavaSpaces. Analiza czasów pobierania i agregacji danych każdego z serwerów potwierdzała fakt, że na całkowity czas obliczeń, po którym wyniki zostaną przedstawione klientowi, ma wpływ czas, po jakim dane zwróci ostatni serwer i czas składania agregatów przez klienta. Czas składania agregatów przez klienta w obu testowanych systemach, przy tych samych danych wejściowych, był w miarę stały. Aby dokładnie uchwycić czasowe różnice między całkowitym czasem obliczeń a czasem zwrócenia agregatów przez ostatni serwer przyjęto wzór:

$$T=T_{całk,i} - \max(T_{serwerAi};T_{serwerBi};T_{serwerCi}),$$

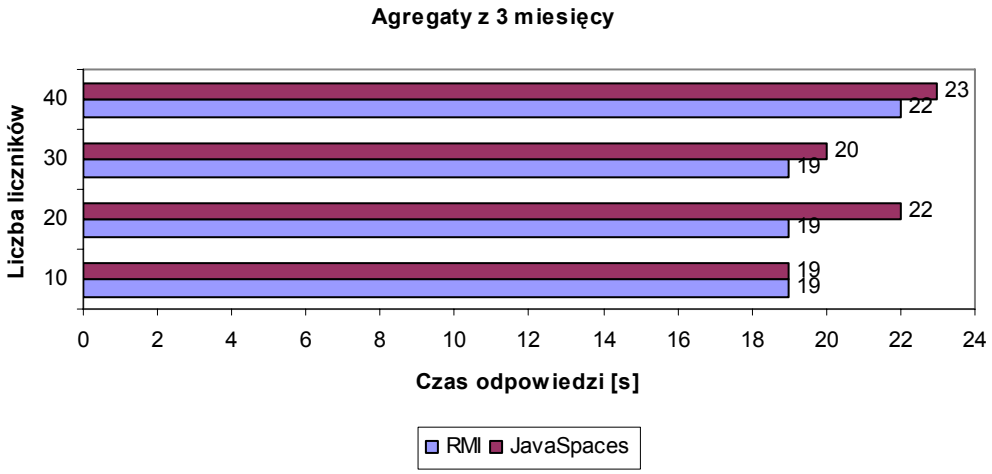
gdzie: $T_{całk,i}$ – całkowity czas, po jakim klient otrzymał wynik, $T_{serwerAi}$; $T_{serwerBi}$; $T_{serwerCi}$ – czasy obliczeń poszczególnych serwerów, i – liczba liczników w danym teście (10,20,30,40).

Przeprowadzenie obliczeń według tego wzoru dla każdego systemu osobno i dla każdego okresu agregacji pozwala dokładnie określić różnicę między metodami komunikacji w tych dwóch systemach. Wyniki analiz przedstawiono poniżej.



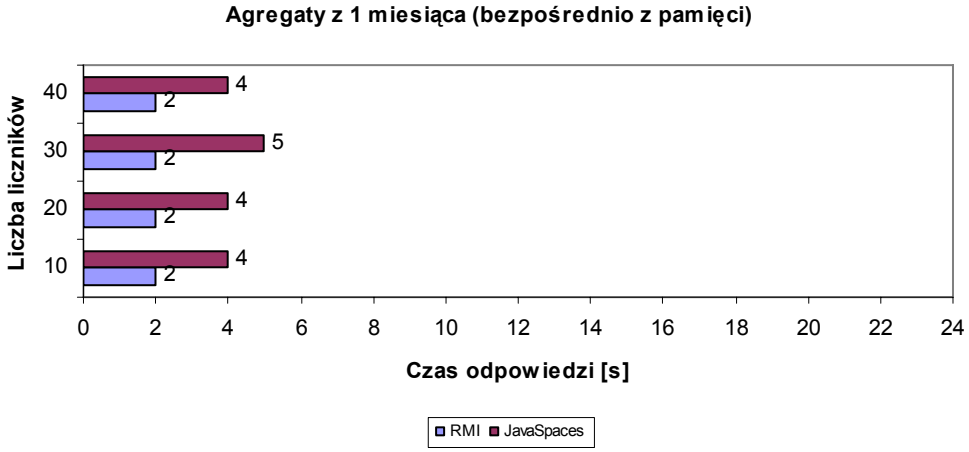
Rys. 7. Różnica całkowitego czasu operacji agregacji i czasu obliczenia danych przez ostatni serwer dla okresu agregacji 1 miesiąc

Fig. 7. Differences between a total time of aggregation process and the longest calculation time from servers. Aggregation period 1 month



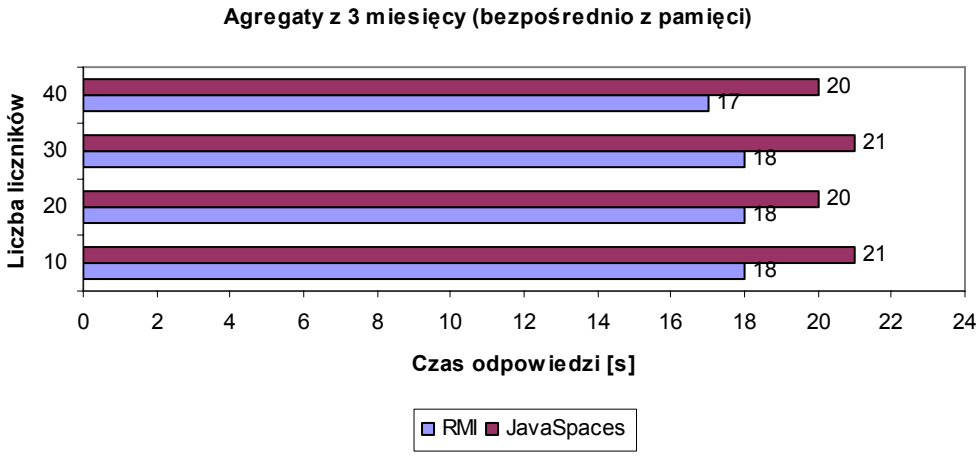
Rys. 8. Różnica całkowitego czasu operacji agregacji i czasu obliczenia danych przez ostatni serwer dla okresu agregacji 3 miesięcy

Fig. 8. Differences between a total time of aggregation process and the longest calculation time from servers. Aggregation period 3 months



Rys. 9. Różnica całkowitego czasu operacji agregacji i czasu obliczenia danych przez ostatni serwer dla okresu agregacji 1 miesiąc

Fig. 9. Differences between a total time of aggregation process and the longest calculation time from servers. Aggregation period 1 month. Data returned from server memory

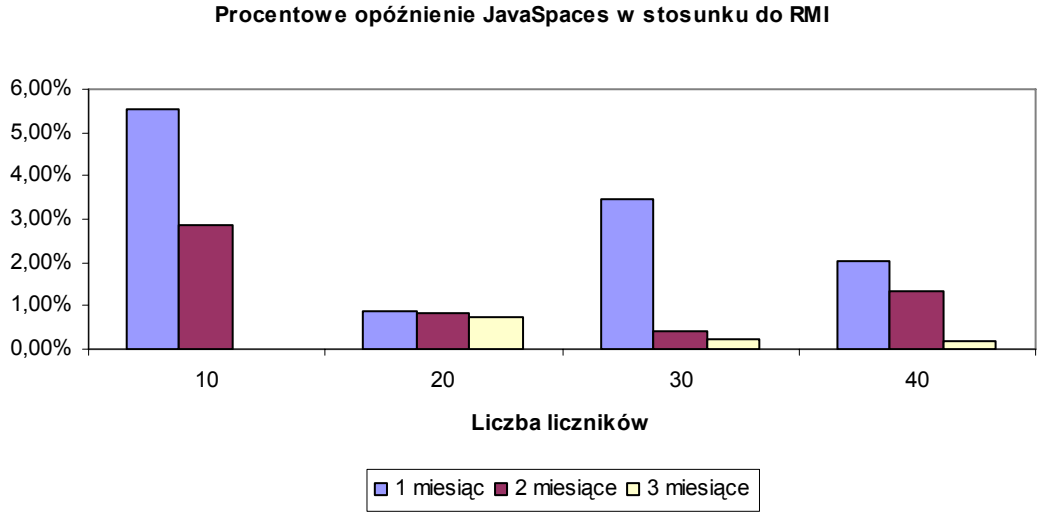


Rys. 10. Różnica całkowitego czasu operacji agregacji i czasu obliczenia danych przez ostatni serwer dla okresu agregacji 3 miesięcy

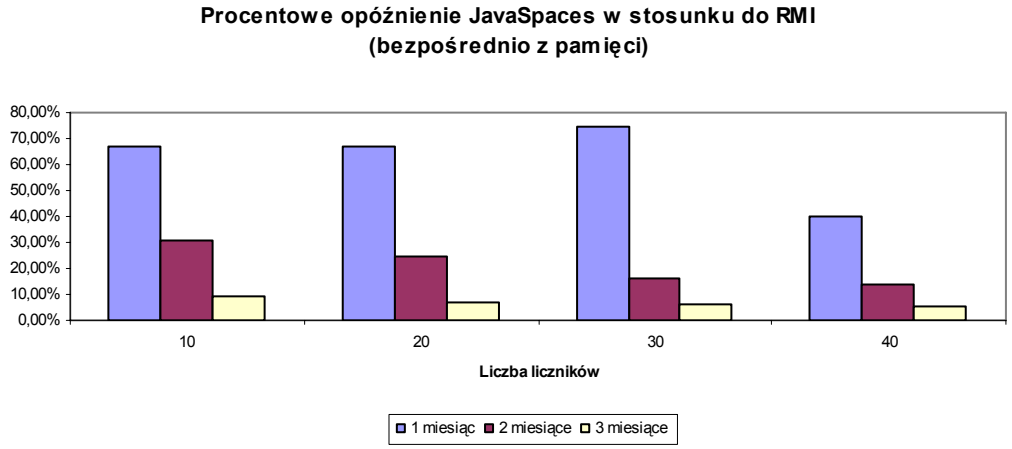
Fig. 10. Differences between a total time of aggregation process and the longest calculation time from servers. Aggregation period 3 months. Data returned from server memory

Na pierwszy rzut oka widać, że RMI jest szybsze (ewentualnie równe - rys.8) w porównaniu z JavaSpaces. Patrząc na testy pierwszego rodzaju widać, że niezależnie od liczby liczników czy okresu agregacji różnica między tymi systemami sięga maksymalnie 4 sekund. Spoglądając na wykresy testów drugiego rodzaju również widzimy, że różnice te są identyczne. Oczywisty jest fakt, że wraz ze wzrostem liczby danych (liczby liczników i okres agregacji) całkowity czas obliczeń rośnie, ale ma on praktycznie niewielki wpływ na różnice czasowe między metodami komunikacji tych dwóch systemów. Ponieważ w tej analizie czas obliczeń ostatniego serwera jest odejmowany od całkowitego czasu całej operacji agregacji,

na całkowity czas obliczeń ma wpływ czas składania agregatów przez klienta i czas komunikacji. Stąd wraz ze wzrostem danych rośnie czas składania agregatów, natomiast czas komunikacji jest stały. Poniżej na wykresach 11 i 12 przedstawiono procentowy wpływ opóźnienia wprowadzonego przez JavaSpaces, które zostało uchwycone w testach poprzednich, w stosunku do całkowitego czasu zebrania agregatów.



Rys. 11. Wpływ opóźnienia serwisu JavaSpaces na całkowity czas operacji agregacji
Fig. 11. JavaSpaces delay compared with a total time of aggregation process



Rys. 12. Wpływ opóźnienia serwisu JavaSpaces na całkowity czas operacji agregacji – dane pobierane bezpośrednio z pamięci
Fig. 12. JavaSpaces delay compared with a total time of aggregation process. Data returned from server memory

W przypadku testów pierwszego rodzaju, dla każdego typu danych wejściowych, opóźnienie serwisu JavaSpaces nie stanowi więcej niż 6% całkowitego czasu obliczeń. Z wykresów tych widać wyraźnie, że przy dużej liczby danych, szczególnie dłuższych okresach agregacji, czas ten spada poniżej 1%.

W testach drugiego rodzaju sytuacja jest nieco inna. Jeśli mamy do czynienia z małą liczbą liczników i krótkim okresem agregacji, wtedy system STDW(RMI) zwraca wyniki niemalże od razu. W takim przypadku kilkusekundowy narzut przy komunikacji w systemie STDW(JINI) istotnie wpływa na całkowity czas otrzymania wyników agregacji. Jednak wraz ze wzrostem okresu agregacji narzut czasowy JavaSpaces w stosunku do całkowitego czasu agregacji gwałtownie spada i przy okresie agregacji z trzech miesięcy stanowi mniej niż 10% całkowitego czasu tej operacji.

6. Podsumowanie

Wyniki testów potwierdzają, że system STDW(RMI) jest szybszy od systemu STDW(JINI), opartego na technologii JINI i serwisie JavaSpaces. Jednakże zaobserwowane różnice prędkości tych systemów są zaledwie kilkusekundowe. Takie różnice czasowe transmisji przy dużej liczbie liczników i długich okresach agregacji są praktycznie pomijalne w stosunku do całkowitego czasu operacji agregacji. Badania wykazały również, że na samą pracę serwerów mają wpływ inne czynniki, których istnienie na tyle zakłóca stabilność pracy serwerów, że różnica między komunikacją RMI a JavaSpaces w rzeczywistych warunkach jest praktycznie niezauważalna. Zatem sens budowania sytemu STDW(JINI) jest teraz oczywisty. Możemy uzyskać wielodostępną (wieloużytkownikową) wersję STDW przy pomijalnych stratach prędkości na komunikacji międzywęzłowej.

Kolejne badania systemu STDW(JINI) związane są z podnoszeniem wydajności architektury wieloużytkownikowej.

LITERATURA

1. Gorawski, M., Malczok, R.: Aggregation and analysis of spatial data by means of materialized aggregation tree. Third Biennial International Conference on Advances in Information Systems, Izmir, Turkey 2004.
2. Gorawski M., Malczok, R.: Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree, 5th Workshop on Spatial-Temporal DataBase Management (STDBM_VLDB'04), Toronto, Canada 2004.
3. Gorawski M., Gabryś M.: Telemetryczny system zintegrowanego odczytu liczników. Praca zbiorowa „Współczesne problemy sieci komputerowych". WNT, Warszawa, 2004, pp. 203- 211.
4. Jini™ Network Technology Specifications v2.0_002.

5. Bill Venners Objects, the Network, and Jini - http://www.artima. com/jini/jiniology /intro.html.
6. Java™ 2 SDK, Standard Edition, Documentation, Sun Microsystems, 2004.
7. Sommers F.: Call on extensible RMI, JavaWorld, December 2003.
8. Freeman E, Hupfer S.: Make Room for JavaSpaces, Part I - an introduction to JavaSpaces, a simple and powerful distributed programming tool, JavaWorld, November 1999.
9. Freeman E.: Make Room for JavaSpaces, Part II - Build a compute server with JavaSpaces, JavaWorld, January 2000.
10. Hupfer S.: Make Room for JavaSpaces, Part III - Coordinate your Jini applications with JavaSpaces, JavaWorld, March 2000.
11. Freeman E, Hupfer S.: Make Room for JavaSpaces, Part IV - Explore Jini transactions with JavaSpaces, JavaWorld, April 2000.
12. Hupfer S.: Make Room for JavaSpaces, Part V - Make your compute server robust and scalable with Jini and JavaSpaces, JavaWorld, June 2000.
13. Hupfer S.: Make Room for JavaSpaces, Part VI -- Build and use distributed data structures in your JavaSpaces programs, JavaWorld, October 2000.
14. Flenner R.: JINI and JavaSpaces Application Development, Sams, 2001.
15. Jan Newmarch Guide to JINI Technologies - http://pandonia.canberra.edu.au.
16. Adam, N., Atluri, V., Yesha, Y., Yu, S. Efficient Storage and Management of Environmental Information, IEEE Symposium on Mass Storage Systems, 2002.

Recenzent: Prof. dr hab. inż. Tadeusz Morzy

Wpłynęło do Redakcji 28 września 2004 r.

Abstract

In this paper we present Spatial Telemetric Data Warehouse which uses Oracle DMBS and JINI technology. We replace the previous way of communication based on standard RMI and we introduce a new way of communication which is modeled as flow of objects. Firstly, we introduce the fundamental ideas and basics concepts used in our system. Next briefly we describe specification of remote method invocation and we present the need for a flow of object model. In Section 2 we present detailed description of JINI specification, we show a major role of lookup service in JINI infrastructure, we introduce a discovery and join process and client's requirements to retreive proxy object from lookup service. JavaSpaces technology specification is introduced in Section 3. We also describe JavaSpaces interface

and present new implementation of remote method invocation model called JERI (Jini Extensible Remote Invocation). Section 4 concerns system details, it includes a detailed description of system components and cascaded star diagram. Finally we compare both models, we present tests, and we show advantages and disadvantages of a new communication model solution .

Adresy

Marcin GORAWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, M.Gorawski@polsl.pl.
Jan PALICA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, mpalica@polsl.gliwice.pl