STUDIA INFORMATICA 2003
Volume 24 Number 2A (53)

Tomasz Jordan KRUK, Mariusz Rafał KSZCZOT
Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej

KOORDYNACJA PRZEZ PRZESTRZENIE KROTEK W ŚRODOWISKACH ROZPROSZONYCH

Streszczenie. W artykule opisano własności komunikacji przez przestrzenie krotek, dokonano wprowadzenia do języków do koordynacji międzyprocesowej. Omówiono technologię Jini i własną propozycję rozszerzenia modelu JavaSpaces.

Słowa kluczowe: krotka, przestrzenie krotek, JavaSpaces.

TUPLE SPACE COORDINATION IN DISTRIBUTED ENVIRONMENTS

Summary. Features of the communication via tuple spaces has been described. Some introductory material related to coordination languages has been mentioned. Both, Jini technology and some extended JavaSpaces model, have been described.

Keywords: tuple, tuple spaces, JavaSpaces.

1. Wstęp

Tworzenie aplikacji równoległych i rozproszonych jest zagadnieniem trudniejszym niż tworzenie aplikacji sekwencyjnych. Głównym problemem jest zapewnienie wymaganej koordynacji między rozproszonymi procesami zrównoleglonej aplikacji. Aby w łatwy sposób wyrażać zależności dotyczące synchronizacji oraz wymiany informacji między procesami, powstało wiele dedykowanych języków do programowania równoległego i rozproszonego. Kompromisem między łatwością korzystania z języka a możliwą do osiągnięcia wydajnością są języki do koordynacji międzyprocesowej. Języki te nie zajmują się opisem obliczeniowej części algorytmu, rozszerzają dowolny język sekwencyjny o możliwość wyrażania synchronizacji i komunikacji międzyprocesowej. Rozproszoną koordynację zapewnia zestaw kilku podstawowych prymitywów, które manipulują krotkami znajdującymi się w przestrzeni

krotek. Idee języków do koordynacji międzyprocesowej nie dają się przenieść bezpośrednio na obiektowy język programowania, jakim jest Java. Implementacja JavaSpaces posiada funkcjonalność języka do koordynacji międzyprocesowej. Technologie Jini i JavaSpaces umożliwiają realizację rozproszonych aplikacji w heterogenicznych sieciach maszyn sekwencyjnych. Model przestrzeni krotek JavaSpaces, zgodny ze specyfikacją przestrzeni krotek Lindy, umożliwia jedynie przekazywanie danych, nie ma natomiast możliwości przekazywania algorytmów, nie rozwiązano również problemu współdzielenia dużych obszarów danych.

2. Przestrzenie krotek - własności komunikacji

Przestrzeń krotek może być postrzegana jako pewna wirtualna pamięć dzielona służąca procesom do koordynacji. Koordynację osiąga się poprzez wykorzystanie kilku podstawowych prymitywów, które manipulują krotkami znajdującymi się w przestrzeni. Każdy prymityw wykonuje pewną akcję na pojedynczej krotce. Krotka (ang. tuple) jest to uporządkowany ciąg danych charakteryzujący się typem i wartością. Każde pole krotki ma ściśle określony typ. Zazwyczaj pole ma również pewną wartość, ale może być również polem określonego typu ale o nieustalonej wartości.

Krotki są adresowane asocjacyjnie. Krotka jest identyfikowana poprzez liczbę jej pól oraz typy danych i wartość poszczególnych pól. Aby odwołać się do pewnej krotki znajdującej się w przestrzeni należy zadać wzorzec o liczbie pól, typach i wartościach równych liczbie pól, typom i wartościom danej krotki. Przy adresowaniu nie ma potrzeby zadawania wartości wszystkich pól - w takiej sytuacji nastąpi odwołanie do krotki o liczbie i typach danych pól jak we wzorcu oraz o wartościach zadanych w polach o określonej wartości. Wartości pól wyspecyfikowanych jako pola o nieokreślonej wartości nie będą rozpatrywane podczas dopasowania.

Ponieważ jedynymi atrybutami krotki są pola charakteryzujące się typem i wartością - dwie równoliczne (co do liczby pól) krotki, których pary odpowiadających pól mają te same typy i te same wartości, są nierozróżnialne.

Istotnymi cechami modelu przestrzeni krotek Lindy są:

- autonomiczność - krotka wstawiona do przestrzeni istnieje niezależnie od procesu, który ją stworzył. Czas życia procesu nie wpływa na czas życia krotki. Proces wstawiający i pobierający krotkę z przestrzeni może działać na logicznie i fizycznie rozłącznej przestrzeni adresowej (ang. time and space decoupling of communication),

- atomowość - krotka znajdująca się w przestrzeni jest elementem niepodzielnym. Krotka nie może być aktualizowana w przestrzeni, w tym celu należy ją pobrać, lokalnie zmodyfikować i następnie ponownie wstawić do przestrzeni. Dostęp do pojedynczej krotki odbywa się na zasadzie wzajemnego wykluczania,
- niedeterminizm – może istnieć wiele krotek pasujących do wzorca zadanego w wywołaniu pobierającym krotkę z przestrzeni. Niezależnie od kolejności ich wstawienia do przestrzeni, nie można jednoznacznie stwierdzić, która z nich zostanie dopasowana,
- równoważenie obciążenia - na poziomie języka nie są widoczne ani miejsce ani sposób przechowywania krotek, do których odwołują się procesy. Równoważeniem obciążenia i zasobów zajmuje się system podtrzymujący wykonywanie procesów.

3. Języki do koordynacji międzyprocesowej

Efektywna koordynacja międzyprocesowa w środowisku rozproszonym może być zrealizowana zgodnie z koncepcją przestrzeni krotek, po raz pierwszy zaproponowaną w postaci języka Linda. Język do koordynacji w środowisku rozproszonym powinien dodatkowo uwzględniać specyfikę otwartego środowiska rozproszonego. Niniejszy rozdział zawiera opis języka C-Linda oraz Ala-C. Język Ala-C rozszerza model koordynacji przez przestrzenie krotek Lindy o wbudowane mechanizmy bezpieczeństwa, możliwość wykorzystania przestrzeni krotek do przekazywania algorytmów między procesami oraz efektywną metodę realizacji dostępu do dużych obiektów dzielonych.

3.1. C-Linda

Język do programowania równoległego w środowisku rozproszonym C-Linda powstał w wyniku połączenia języka do koordynacji międzyprocesowej Linda z obliczeniowym językiem sekwencyjnym C. Składnia języka C-Linda tylko nieznacznie różni się od składni języka C. Język obliczeniowy został rozszerzony o elementy wyrażające koordynację międzyprocesową za pomocą przestrzeni krotek.

W pierwszej implementacji Linda udostępnia trzy prymitywy do operowania na krotkach: out(), in() oraz rd(). Operacja out() pobiera jako argument pola krotki, którą należy stworzyć i następnie wstawić do przestrzeni. Jest to operacja nieblokująca (asynchroniczna), proces wykonujący out() kontynuuje działanie natychmiast po dodaniu krotki do przestrzeni. Argumentami rd() jest zestaw pól tworzących wzorzec adresujący krotkę, której kopię

chcemy uzyskać. Natomiast argumenty in() tworzą wzorzec opisujący krotkę, którą chcemy pobrać (usunąć) z przestrzeni. Zarówno rd(), jak i in() są operacjami blokującymi (synchronicznymi), proces je wywołujący jest wstrzymywany do czasu, aż pasująca do wzorca krotka zostanie wstawiona przez inny proces do przestrzeni i będzie mogła być z niej skopiowana/usunięta. Wywołanie rd() lub in() może pobrać z przestrzeni dowolną krotkę, która spełnia warunek dopasowania (ang. matching). To właśnie dopasowanie konstruuje mechanizm adresowania asocjacyjnego krotek. Wzorzec zadany w wywołaniu jednego z powyższych prymitywów jest porównywany z krotkami znajdującymi się w przestrzeni. Krotka pasuje do wzorca, o ile kolejne pola krotki pasują do kolejnych pól wzorca. Pole krotki pasuje do pola wzorca, o ile ma ten sam typ i albo pole wzorca lub krotki jest nieokreślone co do wartości, albo obydwa pola mają tę samą wartość. Przestrzeń krotek posiada informację o typach poszczególnych pól, jednak nie analizuje w ogóle semantyki informacji przechowywanej w krotkach. To, jaką informację reprezentuje dana krotka zależne jest od konwencji przyjętej przez komunikujące się procesy. Tabela 1 zawiera kilka przykładów wywołań zapisanych przy zastosowaniu składni języka C-Linda.

Tabela 1

Przykłady wywołań operacji na krotkach zapisanych w języku C-Linda

Wywołanie	Opis wywołania
out(„punkt”, 3, 10, 7);	Do przestrzeni wstawiana jest czteroelementowa krotka ("punkt",3,10,7). Może to być na przykład informacja, że pewna funkcja ma w punkcie o współrzędnych (3,10) wartość 7.
int val; in(„punkt”, 3, 10, ?val);	Pobranie z przestrzeni krotek do zmiennej val informacji o wartości funkcji w punkcie (3,10).
int n; in(„globalny”, „licznik”, ?n); n++; out(„globalny”, „licznik”, n);	Wykorzystanie przestrzeni krotek do przechowywania zmiennych globalnych.
in(„uprawnienie”) /* sekcja krytyczna */ out(„uprawnienie”)	Organizacja sekcji krytycznej.

void sema_int(int id, int val); {
for(int i=0; i < val; i++) out(“sem”, id
);
	Pełna implementacja semaforów za pomocą prymitywów Linda. Semafor identyfikowany przez id inicjowany jest wartością val

cd. tabeli 1

} void sema_P(int id) { in("sem", id); } void sema_V(int id) { out("sem", id); }	poprzez wstawienie val identycznych krotek. Następnie, można na nim wykonywać operacje P() i V() poprzez wyjmowanie i ponowne wstawianie krotek do przestrzeni.

3.2. Ala-C

Efektywna koordynacja międzyprocesowa w środowisku rozproszonym może być zrealizowana zgodnie z koncepcją przestrzeni krotek zaproponowaną w języku Linda. Język do koordynacji w środowisku rozproszonym powinien dodatkowo uwzględniać specyfikę otwartego środowiska rozproszonego. Język Ala rozszerza model koordynacji przez przestrzenie krotek o wbudowane mechanizmy bezpieczeństwa, możliwość wykorzystania przestrzeni krotek do przekazywania algorytmów między procesami oraz efektywną metodę realizacji dostępu do dużych danych dzielonych.

Naturalnym rozszerzeniem koncepcji zawartych w języku Linda jest udostępnianie mechanizmów do powoływania i operowania na wielu przestrzeniach krotek. Pojedyncza przestrzeń krotek jest rozwiązaniem akceptowalnym jedynie dla systemów zamkniętych, w których przestrzeń krotek powoływana jest tylko na czas życia konkretnej aplikacji. W systemie otwartym, w którym przestrzeń krotek jest obiektem nasłuchującym na zlecenia od dowolnych procesów, gdyby nie istniała możliwość tworzenia prywatnych, wykorzystywanych przez zamkniętą grupę procesów, przestrzeni krotek, każdy proces mógłby celowo bądź nieświadomie zakłócać przebieg koordynacji innych aplikacji.

W modelu z wieloma przestrzeniami krotek przestrzeń może pełnić nie tylko funkcję medium do wymiany danych między procesami, ale również „agregatu", w którym procesy mogą zbierać krotki o kreślonych własnościach. Linda nie udostępnia jednak mechanizmów informujących, ile jest krotek w przestrzeni krotek pasujących do danego wzorca. Nie wiadomo zatem, ile należy wykonać operacji in(), by pobrać wszystkie krotki, a jednocześnie nie zostać wstrzymanym. Nie można byłoby również korzystać z rd(), gdyż operacja rd() nie gwarantuje, że w każdym wywołaniu będzie pobierać różne krotki pasujące do wzorca. Może się zdarzyć, że kolejne wywołania rd() będą kopiować ciągle tę samą krotkę. Aby rozwiązać ten problem i zwiększyć elastyczność modelu, zdefiniowano dodatkowe funkcje, operujące na kolekcji krotek. W języku Ala-C pojawiają się dodatkowe prymitywy: collect(), copy(), clear(), które umożliwiają odpowiednio przeniesienie, skopiowanie i usunięcie kolekcji krotek pasujących do zadanego wzorca.

3.3. Krotki aktywne

Język do koordynacji międzyprocesowej powinien udostępniać metody tworzenia nowych procesów. Język Linda udostępnia prymityw eval(), umożliwiający tworzenie procesu uruchamianego w ramach przestrzeni, język Ala definiuje modyfikatory pól krotek: eval oraz interface. Krotki zawierające algorytm noszą miano krotek aktywnych, w odróżnieniu od zwykłych krotek pasywnych, zawierających jedynie statyczne dane.

Operacja ewaluacji pola krotki wykonywana jest przez przestrzeń krotek, a nie przez proces wstawiający krotkę. W konkretnych realizacjach ewaluacja może się sprowadzić do utworzenia przez przestrzeń krotek osobnego procesu (bądź wątku sterowania) dedykowanego do wyznaczenia wartości danego pola. Do czasu zakończenia wszystkich ewaluacji krotka nie podlega dopasowaniom do wzorców innych operacji na krotkach (np. skopiowania czy pobrania), jest dla nich niewidoczna. Procesy mogą wykonywać kod wyrażeń aktywnych języka obliczeniowego (zwykle funkcji). Istotnym ograniczeniem jest niemożność tworzenia procesów wykonujących się zgodnie z algorytmem funkcji zdefiniowanej w innym uczestniczącym w koordynacji procesie. Mechanizm eval nie umożliwia realizacji zdalnego wywołania procedury.

Operacja interface, dostępna jedynie w języku Ala, umożliwia interpretację danego pola krotki jako nazwy pewnego wyrażenia aktywnego (np. funkcji czy metody obiektu) języka obliczeniowego. W miejsce pola do przestrzeni krotek wstawiany jest algorytm wyznaczania wyrażenia. Samo wstawienie krotki nie powoduje wykonania się algorytmu, nie powoduje go również usunięcie krotki z przestrzeni. Jedynie wykonanie operacji rd() na krotce z polem zawierającym algorytm powoduje utworzenie nowego procesu, który wykonuje algorytm.

Krotka z polem zawierającym algorytm będzie pasowała do wzorca zadanego w operacji in()/rd(), jeżeli odpowiednie pole wzorca będzie zawierało wywołanie interfejsu o zwracanym typie oraz liczbie i typach argumentów takich, jak zwracany typ oraz liczba i typy argumentów wstawianego interfejsu. Przekazywanie algorytmów służy w rzeczywistości do realizacji zdalnego wywołania procedury.

Więcej informacji na temat koordynacji międzyprocesowej poprzez przestrzenie krotek w językach C-Linda oraz Ala-C można znaleźć w pozycjach [1, 2].

4. Środowisko rozproszone na maszynach wirtualnych Javy

Technologie Jini i JavaSpaces umożliwiają tworzenie oprogramowania rozproszonego w systemach, które z natury są jedynie sieciowymi systemami operacyjnymi.

4.1. Architektura Jini

Jini to projekt badawczy firmy Sun Microsystems, który stanowi elastyczną architekturę udostępniania zasobów w sieci. Celem twórców Jini jest maksymalne ułatwienie konstrukcji systemów sieciowych opartych na klasach Javy. W sieci Jini uczestnikami komunikacji są ludzie, maszyny, programy, dane, urządzenia – słowem, cokolwiek zdolnego do oferowania usług bądź pragnącego skorzystać z usług innych uczestników. Jini oparte jest na idei sieci spontanicznych, do których podłączenie się i wykorzystanie ich zasobów odbywa się bez wcześniejszego planowania instalacji czy interwencji użytkownika. Po podłączeniu do sieci nowe urządzenie zgodne z Jini rozgłasza komunikat sygnalizując swoją obecność. Z odpowiedzi dowiaduje się o innych urządzeniach. Usługa wyszukiwania rejestruje nowe urządzenie, zatrzymuje rekord jego atrybutów i wysyła mu swoją lokalizację. Gdy na przykład zachodzi potrzeba skorzystania z pewnej usługi, urządzenie odwołuje się do usługi wyszukiwania, znajduje to, co niezbędne, i wysyła rezultaty do odpowiedniego urządzenia.

Jini to zbiór różnych API oraz protokołów, które mogą pomóc w tworzeniu, a następnie rozwoju systemów rozproszonych, zorganizowanych jak federacje usług. Usługa - urządzenie, oprogramowanie, kanał komunikacyjny - może rezydować w sieci, gotowa do wykonania przydatnej funkcji. Federacja usług to zbiór dostępnych usług, które klient (program, usługa lub użytkownik) może wykorzystać w jakimś określonym celu. Jedną ze standardowych usług Jini jest technologia JavaSpaces, która jest pochodzącą od Sun Microsystems specyfikacją przestrzeni krotek Lindy dla języka Java.

4.2. Usługa JavaSpaces

JavaSpaces jest technologią ściśle związaną z Jini, ponieważ wprowadza nowy model przetwarzania rozproszonego. JavaSpaces zostało stworzone jako serwis Jini. Jest transakcyjna i prosta w użyciu. Wprowadza nowe możliwości komunikacji między obiektami w Jini.

Wśród założeń, które przyświecały twórcom JavaSpaces, należy zwrócić szczególną uwagę na kilka. JavaSpaces ma służyć współpracującym w ramach grupy programom działającym współbieżnie w ramach środowiska rozproszonego. Zakłada ponadto anonimowość pomiędzy aplikacjami, brak bezpośredniej komunikacji między nimi, lecz w zamian komunikacje niezależną od czasu i miejsca, przy użyciu prostych mechanizmów synchronizacji i komunikacji na wysokim poziomie.

Przestrzenie JavaSpaces posiadają kilka zasadniczych cech, które czynią je tak potężnym narzędziem, a mianowicie:

- JavaSpaces są dzielone - wiele zdalnych procesów może z nich korzystać współbieżnie. Przestrzeń sama zajmuje się kontrolą współbieżnego dostępu.
- JavaSpaces są trwałe (ang. persistent) - przestrzenie dostarczają niezawodnych mechanizmów przechowywania obiektów.
- JavaSpaces są asocjacyjne - obiekty są wyszukiwane w przestrzeniach na zasadzie skojarzenia, a nie położenia w pamięci czy też identyfikatora. Powoduje to, że obiekty są wyszukiwane na podstawie ich zawartości bez konieczności brania pod uwagę, jaki konkretny obiekt jest wyszukiwany, kto go stworzył lub gdzie jest przechowywany.
- JavaSpaces są transakcyjne - przestrzenie używają mechanizmów transakcji, aby zapewnić, że operacja na przestrzeni jest atomowa (niepodzielna).

JavaSpaces definiuje pięć podstawowych operacji, które są dostarczane przez każdy obiekt implementujący interfejs JavaSpaces. Są to standardowe (znane ze specyfikacji Lindy) blokujące operacje odczytu/zapisu/pobrania (read()/write()/take()) i dodatkowe nieblokujące ich odmiany (readIfExist(), takeIfExist()). Ponadto, w modelu JavaSpaces zdefiniowany został mechanizm powiadomień (ang. notify), który umożliwia realizację powiadomienia o zapisywanych krotkach do przestrzeni, pasujących do wyspecyfikowanego wzorca.

Istotnym ograniczeniem JavaSpaces jest brak możliwości wykonywania algorytmów przez samą przestrzeń. Nie ma dostępnego mechanizmu ewaluacji pola krotki, jak również brak jest mechanizmu interfejsów znanego z języka Ala-C. Wprawdzie JavaSpaces przechowuje obiekty wraz ze wszystkimi metodami, jednak wywołanie metody obiektu jest możliwe dopiero po wyjęciu go z przestrzeni. Obiekty znajdujące się w przestrzeni JavaSpaces są jedynie statycznymi danymi. W JavaSpaces pojawia się również problem nieefektywnego dostępu do dużych obszarów danych. Nie został stworzony mechanizm dostępu referencyjnego. Problemy te powodują, że Jini i JavaSpaces nie są optymalnymi narzędziami do tworzenia aplikacji rozproszonych.

Więcej na temat technologii Jinii oraz JavaSpaces można znaleźć w pozycjach [5, 6] spisu literatury.

5. AlaSpaces - koncepcja rozszerzenia modelu JavaSpaces

Koncepcja o nazwie AlaSpaces rozszerza przestrzeń krotek JavaSpaces o mechanizmy krotek aktywnych oraz dostępu referencyjnego do dużych obszarów danych. Zaproponowana została dodatkowa usługa technologii Jini, rozszerzająca model przestrzeni

JavaSpaces o nowe mechanizmy. Usługa została nazwana AlaSpaces, ponieważ łączy w sobie funkcjonalność języka Ala i technologii JavaSpaces.

AlaSpaces jest dodatkową warstwą pomiędzy aplikacją rozproszoną a przestrzenią krotek JavaSpaces. Interfejs programisty AlaSpaceAPI usługi AlaSpaces udostępnia identyczne pod względem semantycznym metody z JavaSpaces, jedynie operujące na nieco innej klasie obiektów reprezentujących krotki. Taka organizacja umożliwia łatwą migrację oprogramowania z JavaSpaces do AlaSpaces. Oprogramowanie napisane z wykorzystaniem technologii JavaSpaces będzie nadal działać w tym samym środowisku rozproszonym, co oprogramowanie napisane z wykorzystaniem AlaSpaces.

5.1. Tworzenie procesów

Jini wraz z mechanizmem RMI (leżącym u podstaw Jini) dostarcza wszystkich potrzebnych elementów, umożliwiających implementację krotek aktywnych. Przesłanie kodu algorytmu zawartego w krotce aktywnej realizowane jest poprzez wewnętrzne mechanizmy usługi AlaSpaces. Proces ściągnięcia kodu jest całkowicie niewidoczny dla użytkownika. Informacja o miejscu, z którego należy pobrać kod, nazwa klasy i metody oraz ewentualne parametry wywołania zapisane są w odpowiednich polach krotki. Proces wykonujący algorytm zostanie utworzony i wykonany z wykorzystaniem zasobów komputera, na którym uruchomione jest środowisko Jini. Krotka nie będzie widoczna w przestrzeni (nie będzie podlegać dopasowaniom) dopóki algorytm nie zakończy się, dopiero wtedy będzie możliwe wyjęcie krotki z przestrzeni i odczytanie rezultatu wykonania algorytmu. Aby krotka stała się krotką aktywną, należy wywołać metodę SetEvaluate() lub SetInterface() obiektu reprezentującego krotkę, jeszcze przed wpisaniem jej do przestrzeni. Nazwę klasy, metody oraz parametry wywołania umieszcza się wywołując metodę InsertCode() obiektu krotki.

5.2. Realizacja dostępu referencyjnego do dużych obszarów danych

W środowisku AlaSpaces dostęp referencyjny można uzyskać jedynie do obiektów klasy ARefArray. Klasa ARefArray jest na tyle elastyczna, że za jej pomocą można tworzyć obiekty reprezentujące tablice o dowolnym wymiarze, w skład których mogą wchodzić obiekty o dowolnym typie. Tworząc nowy obiekt reprezentujący tablicę referencyjną, należy wywołać konstruktor klasy ARefArray z parametrem określającym jej wymiar. Aktualny indeks pola tablicy, na którym będą przeprowadzane operacje, ustala się wywołując metodę SetIndex() tyle razy, ile wynosi wymiar tablicy. Dostęp do wartości pola umożliwiają metody get() - pobranie wartości oraz set() - ustalenie wartości. Zapis tablicy referencyjnej do krotki dokonuje się przez wywołanie metody InsertReference(). Przed wstawieniem krotki

referencyjnej do przestrzeni należy wcześniej wywołać metodę SetReference() dla jawnego zaznaczenia, że dostęp do krotki będzie odbywał się w sposób referencyjny. Jeżeli metoda ta nie zostanie wywołana, krotka będzie traktowana jak zwykła krotka pasywna. Od chwili wstawienia krotki z dostępem referencyjnym do przestrzeni, każdy proces dokonujący zmian na polach tablicy operuje jedynie na jej prywatnej kopii. Synchronizacja i zapisanie wprowadzonych zmian do oryginalnej tablicy następuje, gdy dowolny z procesów operujących na tej samej krotce referencyjnej wywoła metodę Synchronize() klasy AlaSpace.

Więcej szczegółów na temat implementacji AlaSpaces można znaleźć w pozycji [3] literatury.

LITERATURA

1. Kruk T. J.: Komunikacja przez przestrzenie krotek w rozproszonych systemach operacyjnych. Konferencja POLMAN' 99, s. 112-127. Poznań 1999.
2. Kruk T. J.: Komunikacja międzyprocesowa w rozproszonych systemach operacyjnych. Praca doktorska, Politechnika Warszawska, Warszawa 1999.
3. Kszczot M. R.: Język *Ala-Java* – rozszerzenie JavaSpaces o funkcjonalność języka Ala. Praca dyplomowa, Politechnika Warszawska, Warszawa 2003.
4. Halter S. L.: JavaSpaces Example by Example. The Sun Microsystems Press 2000.
5. JavaSpaces Service Specification. Sun Microsystems. Dec. 2001.
6. Jini Technology Core Platform Specification. Sun Microsystems. Dec. 2001.
7. A Collection of Jini Technology Helper Utilities and Services Specifications. Sun Microsystems. Dec. 2001.
8. Jini Device Architecture Specification. Sun Microsystems. Dec. 2001.
9. Jini Network Technology - An Executive Overview. Sun Microsystems. Feb. 2001.

Recenzent: Dr inż. Hafedh Zghidi

Wpłynęło do Redakcji 11 kwietnia 2003 r.

Abstract

Coordination languages are based on the idea of tuples and tuple space. They enable temporal and referential uncoupling of processes. Structures which enable communication

and synchronization of processes are called tuples. Each tuple consists of a sequence of fields. Each field is always typed and either has a value or is uninitialized. Tuples may be inserted into tuple space by one process and taken/ removed from the space by other process. When a process tries to remove a tuple this tuple is addressed associatively, namely the tuple from the space matches given pattern if it contains the same amount of typed fields and initialized field values match the field values given in the pattern.

Coordination language concepts have been introduced in the language Linda and its implementation C-Linda, which has slightly enhanced ANSI C language. New enhancements in the idea of coordination languages have been proposed by means of a new language, Ala, and its Ala-C implementation. Meanwhile tuple space idea has been implemented in the Java technologies group, namely in Jini as its subpart called JavaSpaces. Authors describe proposed and implemented prototype of some enhancements to JavaSpaces taken from the Ala language concepts.

Adresy

Tomasz Jordan KRUK: Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa, Polska, T.Kruk@ia.pw.edu.pl .

Mariusz Rafał KSZCZOT: Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej, ul. Nowowiejska 15/19, 00-665 Warszawa, Polska, M.Kszczot@elka.pw.edu.pl