12.

Agenci programowi jako brokerzy zasobów w Gridzie

**Wojciech Kuranowski, Maria Ganzha,
Marcin Paprzycki, Mateusz Dominiak**

12.1. Wprowadzenie

Wykorzystanie zasobów dostępnych w sieciach typu Grid[1] jest bardzo obiecującym podejściem do stworzenia nowej infrastruktury obliczeniowej. Umożliwi ona (infrastruktura) nie tylko wykorzystanie mocy obliczeniowej różnorodnych, rozproszonych geograficznie komputerów, ale również możliwość wykorzystania zasobów na zasadzie opłaty za rzeczywiste ich wykorzystanie (np. zamiast kupowania i instalowania lokalnie licencji na oprogramowanie, które jest wykorzystywane tylko sporadycznie). Nie ma wątpliwości, że rozwiązania gridowe zaczynają wpływać na naukę, biznes i przemysł. Jednakże ich akceptacja, mimo dającego się zauważyć przyspieszenia, nie jest satysfakcjonująca. Jednym z możliwych powodów takiego stanu rzeczy jest nadmierne skomplikowanie oprogramowania obsługującego udostępnianie zasobów i zarządzanie nimi w Gridzie.

Równocześnie pojawiły się publikacje sugerujące, że zastosowanie w Gridzie ontologii, oraz agentów programowych może doprowadzić do przełomu przez dostarczenie sieciom Grid potrzebnej inteligentnej infrastruktury [6, 15]. Zgadzając się z zaprezentowanymi tam argumentami, przeszukaliśmy istniejące rozwiązania, wprowadzające do Gridu agentów programowych. Podsumowanie znalezionych rozwiązań znajduje się w następnym podpunkcie, jednak już teraz można powie-

[1] Według najnowszych wytycznych Unii Europejskiej, należy pisać Grid z wielkiej litery.

dzieć, że wszystkie one są ograniczone jeśli chodzi o zakres oraz nie rozwiązują pewnych istotnych problemów. Dlatego przedstawiamy własne rozwiązanie bazujące na zespołach agentów, które współpracując ze sobą, wypełniają zadania stawiane przez klienta. Następnie opiszemy sposób efektywnej implementacji usługi odnajdywania zasobów w Gridzie. Na zakończenie omówimy procedury wybierania zespołu przez agenta do wykonania obliczeń, wybór zespołu do przyłączenia się przez agenta oraz kierunki rozwoju naszych dalszych badań.

12.2. Agenci w Gridzie – stan badań

Jedną z najwcześniejszych prac dotyczących agentów w Gridzie jest artykuł J. Cao i współpracowników, którzy podjęli temat odnajdywania zasobów w Gridzie [2]. Zaproponowali oni hierarchiczną strukturę agentów i opracowali różnorodne strategie optymalizacyjne dla dystrybucji informacji. Mimo interesującego rozwiązania, skupiono się w nim tylko na bardzo wąskim fragmencie zastosowań agentów w Gridzie. Dodatkowo zaproponowane podejście bazowało na infrastrukturze PACE, która (na ile byliśmy to w stanie stwierdzić) nie jest już obecnie rozwijana.

Jedną z późniejszych, i bardzo istotnych, propozycji był system MAGDA (Mobile AGent Distributed Application) zaprojektowany przez B. Di Martino i O. Rana [12]. Był to system agentowy zbudowany dla (1) odnajdywania zasobów w Gridzie, (2) monitorowania i balansowania obciążeń poszczególnych węzłów Gridu, oraz (3) wykonywania zadań w Gridzie. Dedykowani mobilni agenci odwiedzali serwery w Gridzie i gromadzili informacje (zbierane przez lokalnych statycznych agentów) używane później do optymalizacji dystrybucji obciążenia generowanego przez aplikacje. Dzięki temu możliwe było podejmowanie takich decyzji, jak przeniesienie wykonywanego zadania z obciążonego hosta na mniej obciążony (analiza danych odbywała się za pomocą programu transportowanego przez agenta). Zaproponowane rozwiązanie nie miało jednak związanego z nim modelu ekonomicznego (czyli: kto? komu? płaci, i za co?). Ponadto system MAGDA zaimplementowany był za pomocą platformy Aglets, która od roku 2001 nie jest już rozwijana.

W roku 2005 S. S. Manvi ze współpracownikami zaproponowali inne podejście do agentów w Gridzie [9]. Swoje rozważania rozpoczęli od modelu ekonomicznego, w którym mobilni agenci przemieszczają się w sieci w celu wykonania zadania użytkownika. W każdym odwiedzonym miejscu, agent dowiaduje się jakie są lokalne warunki dla wykonania zadania i jeśli są akceptowalne, może zlecić jego wykonanie, jeśli nie, przemieszcza się dalej. Autorzy przedstawili algorytmy wyboru kolejnego węzła do odwiedzin. Jednakże nie wzięli oni pod uwagę dwóch istotnych problemów. Po pierwsze tego, jaki jest koszt przemieszczania się

w Gridzie agenta niosącego ze sobą zadanie oraz wykonanie go. Po drugie, co się stanie jeśli host, w którym wykonywane jest zadanie przestanie funkcjonować (na przykład, z powodu awarii zasilania).

Również w 2005 roku, D. Ouelhadj ze współpracownikami rozważał negocjacje (i renegocjacje) SLA (Service Level Agreement) między agentem zarządzającym zasobami, a agentem chcącym je wykorzystać [10]. Negocjacje były przeprowadzane za pomocą protokołu Contract Net Protocol. Jednakże ich praca skupiała się wyłącznie na negocjacjach dotyczących SLA wprowadzając hierarchię planistów i meta-planistów.

Mimo tego, że opisane rozwiązania są bardzo interesujące, nie są one wolne od problemów. (1) Większość z nich ma ograniczony zakres i funkcjonalność, a ponadto nie bierze pod uwagę kwestii ekonomicznych związanych z zapłatą za korzystanie z zasobów. (2) Niektóre z nich bazowane są na mobilności agentów, nie biorąc pod uwagę kosztu transportu kodu i danych potrzebnych do wykonania zadania. (3) Nie biorą one pod uwagę bardzo dynamicznej natury Gridu i wykorzystują pojedynczych usługodawców. Sprawia to, że klienci narażeni są na potencjalne problemy związane z nagłym i bez ostrzeżenia znikaniem i pojawianiem się (czy też skokowymi zmianami obciążenia) poszczególnych zasobów. (4) Poleganie na „mało znanych" („nieznanych") dostawcach usług powinno pociągnąć za sobą zaproponowanie procedur zarządzania zaufaniem/reputacją.

12.3. Proponowane rozwiązanie

Zacznijmy od tego, że w naszych rozważaniach traktujemy Grid jako „otwarte środowisko", do którego może przyłączyć się dowolny komputer podłączony do Internetu. W ten sposób wracamy do oryginalnej wizji przedstawionej w pracy [7]. Lokalne Gridy, których zasięg ogranicza się do laboratorium, bądź też firmy, pozwalają na dodatkową kontrolę wykorzystania zasobów, oraz ingerencję administratora w sytuacji problemowej. Pozwala to na pominięcie problemów, które niesie ze sobą otwarty Grid i sprawia, że są one mniej ciekawe. Po drugie, w Gridzie widzimy środowisko, w którym pracownicy (pojedynczy agenci, reprezentujący indywidualnych internautów) chcą być wynagrodzeni za użyczone zasoby. W tym celu wchodzą oni w interakcje z użytkownikami, którzy chcą wykorzystać oferowane w Gridzie usługi obliczeniowe (i są gotowi za to zapłacić).

W tym kontekście łatwo jest zauważyć, że pojedynczy pracownik, reprezentujący typowego domowego użytkownika, ma raczej ograniczoną wartość. Projekt SETI@home, który zgromadził miliony domowych komputerów odniósł niewątpliwy sukces (w pewnych okresach stanowił zasób obliczeniowy kilkakrotnie potężniejszy niż największy istniejący w tym czasie superkomputer), jednak zadanie obliczeniowe rozwiązywane w tym projekcie pozwalało na to, że zasoby mogły

znikać w dowolnym momencie, aby powrócić równie nieoczekiwanie. Ponadto, obliczenia mogły być wykonane w dowolnej kolejności oraz w dowolnym czasie. Dla porównania należy podkreślić, że obliczenia „biznesowe" najczęściej muszą być zakończone w określonej kolejności, a także zazwyczaj w zdefiniowanym czasie. W takich aplikacjach wymagana jest pewna forma SLA (Service Level Agreement). Równocześnie zagwarantowanie SLA w przypadku domowych komputerów jest praktycznie niemożliwe (nie daje się przewidzieć kiedy pies wyciągnie wtyczkę z kontaktu, lub kiedy właściciel zapomni o liczonym zadaniu i zacznie grać w Quake IV).

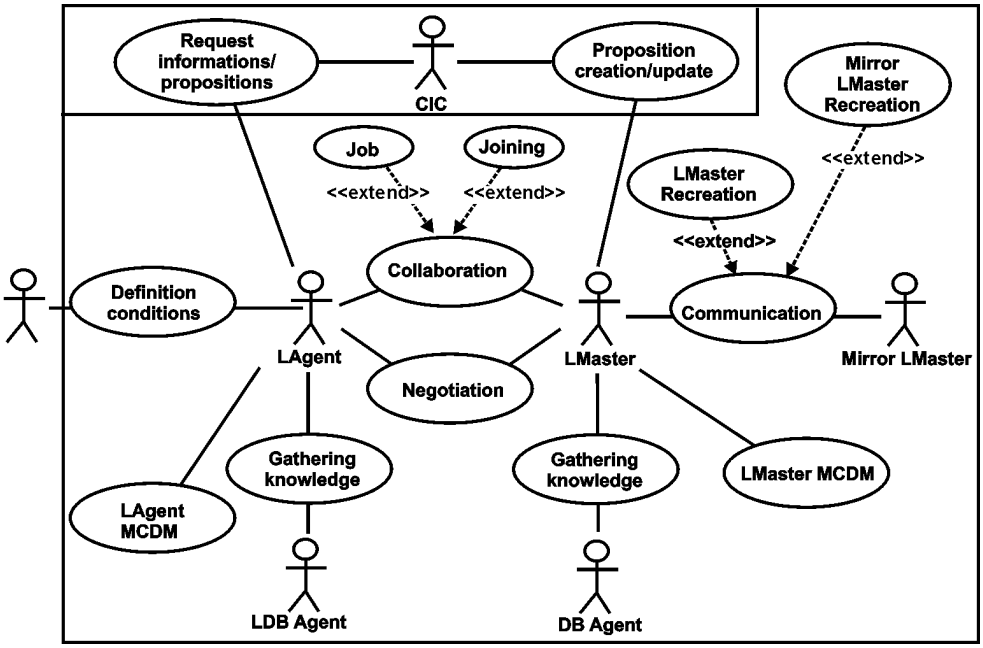
W celu rozwiązania zarysowanych problemów, proponujemy zastosowanie zespołów współdziałających agentów, których działanie oparte jest na kilku podstawowych założeniach:

- agenci pracują w zespołach (grupach agentów),
- każdy zespół ma swojego lidera – agent LMaster,
- każdy LMaster posiada agenta LMirror, który może przejąć jego funkcję w przypadku awarii,
- Pracownicy przyłączają się do zespołów według indywidualnych kryteriów,
- zespoły (reprezentowane przez LMasterów) akceptują chcących przyłączyć się Pracowników na podstawie własnych kryteriów,
- decyzje dotyczące przyłączenia się i akceptacji podejmowane są na podstawie analizy wielokryterialnej,
- każdy agent Pracownik może pełnić rolę LMirrora (czyli również LMastera),
- dopasowywanie (*matchmaking*) umożliwia usługa *Yellow Pages* udostępniana przez agenta CIC.

Diagram 12.1 przypadków użycia przedstawia zaprezentowane założenia w postaci całościowego obrazu systemu.

Aby zespół był widoczny dla każdego potencjalnego użytkownika, informacja o nim musi być łatwo dostępna. Istnieje wiele sposobów na udostępnienie takiej informacji w rozproszonych środowiskach, a każdy z nich ma swoje wady i zalety [16]. W naszym projekcie zdecydowaliśmy się użyć metody *Yellow Pages*. Dokładnie rzecz biorąc, zaproponowaliśmy stworzenie centralnego repozytorium ofert, tzw. centrum informacyjne (*client information center*; CIC). Publikowane tam są takie dane, jak posiadane przez zespół zasoby (np. charakterystyka dostępnego sprzętu, oprogramowanie) czy też informacja, że dany zespół szuka agentów Pracowników (i ich charakterystyka), oraz metadane (np. adres LMastera). Yellow Pages może być używane zarówno w przypadku, gdy (1) agenci użytkownika szukają potrzebnych do wykonania obliczeń zasobów, jak i (2) agenci Pracownicy szukają zespołu do przyłączenia się. Na przykład, Pracownik reprezentujący kom-

puter z oprogramowaniem Maple, może szukać zespołu, który specjalizuje się w rozwiązywaniu zadań z użyciem Maple.



Rysunek 12.1. Diagram przypadków użycia

Źródło: opracowanie własne.

Załóżmy teraz, że proponowany system działa już od pewnego czasu, przez co powstała już grupa zespołów, a informacje o nich (zarówno posiadane zasoby, jak i warunki przyłączenia dodatkowych agentów) opublikowane zostały za pośrednictwem CIC. Użytkownik zaprezentowany na rysunku 12.1 może chcieć przyłączyć się do zespołu lub wykorzystać zasoby oferowane przez jeden z zespołów (zauważmy, że ten sam użytkownik pewnego dnia może chcieć zaoferować swoje zasoby, a innego skorzystać z zasobów, których nie posiada). Oba te procesy mogą być zamodelowane symetrycznie.

Użytkownik chcący przyłączyć się do zespołu definiuje warunki, które muszą być spełnione przy przyłączeniu i komunikuje je do swojego agenta (LAgenta, który ma stać się agentem Pracownikiem). Należy zauważyć, że użytkownik może również zlecić stworzenie nowego zespołu, w którym LAgent stanie się LMasterem, jednakże tą sytuacją nie będziemy się zajmować. LAgent komunikuje się z CIC, aby otrzymać listę zespołów, które spełniają zdefiniowane wymagania. Na podstawie poprzednich doświadczeń (jeśli istnieją) odfiltrowuje zespoły, z którymi

współpraca nie przebiegała pomyślnie; np. nie dostał uzgodnionego wynagrodzenia (informacja o podobnym schemacie zachowań znajduje się w pracy [1] oraz w rozdziale 11). Do pozostałych LMasterów reprezentujących poszczególne zespoły wysyłany jest komunikat zgodny z protokołem FIPA Contract Net, który rozpoczyna proces negocjacji i analizy otrzymanych ofert. Interakcje między LAgentem a LMasterami mogą zakończyć się dwojako: (1) znaleziono zespół, który spełnił oczekiwania i przyłączono się do niego, (2) nie otrzymano żadnych ofert, albo żadna z otrzymanych ofert nie była dostatecznie atrakcyjna. W takim momencie LAgent informuje użytkownika o tym fakcie i oczekuje na dalsze instrukcje.

W momencie, gdy użytkownik chce wykonać obliczenia w Gridzie, również definiuje wymagania, które muszą zostać spełnione, np. maksymalną cenę, dostępne oprogramowanie, etc. LAgent odpytuje CIC, aby poznać listę zespołów, którzy mogą wykonać jego zadanie. Tutaj następuje pierwsze filtrowanie ofert, albowiem zespoły, które nie spełniają, np. wymagań sprzętowych, nie będą włączone do listy przesłanej przez CIC. Podobnie, jak w poprzednim scenariuszu, z listy usuwane są zespoły niegodne zaufania, np. które poprzednio nie wywiązały się z ustaleń. Następuje komunikacja z pozostałymi zespołami za pomocą protokołu FIPA Contract Net, w trakcie której wybierany jest najlepszy zespół do wykonania danego zadania. Jeśli żaden z zespołów nie przedstawi oferty spełniającej określone przez użytkownika wymagania (lub jeśli nie będzie żadnej oferty), LAgent raportuje to użytkownikowi i oczekuje na dalsze instrukcje.

Następną ważną kwestią systemu jest powiązanie LMastera z LMirrorem. W momencie, gdy tworzony jest nowy zespół, LAgent który go założył zostaje LMasterem. Pierwszy agent, który przyłączy się do zespołu zostaje LMirrorem (agentem, który może przejąć rolę lidera zespołu w przypadku awarii LMastera). Kolejni agenci, którzy przyłączą się do zespołu zostają Pracownikami. Nie zdecydowaliśmy jeszcze czy LMirror powinien również pełnić rolę Pracownika, czy też jedynie replikować zasoby LMastera. Decyzja ta zostanie podjęta po eksperymentach mających na celu zbadanie obciążenia LMirrora generowanego podczas replikacji. LMaster i LMirror dzielą wszystkie informacje, które są niezbędne do istnienia zespołu, np. listę Pracowników, listę wykonywanych zadań, historię przeprowadzonych negocjacji. Monitorują się również wzajemnie w krótkich odstępach czasu. W przypadku, kiedy LMaster przestaje odpowiadać na wiadomości ACL wysyłane przez LMirrora (ping), LMirror kontaktuje się z agentem odpowiadającym za infrastrukturę całego środowiska agentowego (Directory Facilitator) odpytuje go o stan LMastera. Jeśli nie znajduje go tam, LMirror przejmuje jego rolę. Informuje wszystkich członków zespołu, że nastąpiła zmiana LMastera. Następnie wybierany jest nowy LMirror, na którego replikowany jest nowy LMaster. Podobnie, gdy LMaster stwierdzi, że LMirror uległ awarii, spośród innych członków zespołu wybierze nowego LMirrora i dokona replikacji. Zaproponowane rozwiązanie nie jest bez wad. Możemy sobie wyobrazić, że LMaster i LMirror ulegają awa-

rii w tym samym czasie, co spowoduje, że zespół przestanie istnieć. Jednakże takie sytuacje będą relatywnie rzadkie, a naszym celem nie jest stworzenie całkowicie odpornej na wszelkie awarie infrastruktury, ale infrastruktury zapewniającej bezawaryjność na rozsądnym poziomie. Oczywiście w środowisku produkcyjnym powstałyby dodatkowe mechanizmy zapobiegające destrukcji zespołu.

Kolejny element przedstawiony na rysunku 12.1 to funkcja Gathering knowledge. Określa ona fakt, że w systemie gromadzone są informacje dotyczące jego funkcjonowania. LMaster przechowuje informacje o wszystkich interakcjach z LAgentami, (jako przedstawicielami użytkowników pragnących wykorzystać jego zasoby), jak i z członkami swojego zespołu (Pracownikami i LMirrorem). Dzięki zgromadzonym danym może on podejmować decyzje dotyczące, np. wykluczenia interakcji z określonymi agentami. Podobnie LAgent zbiera wiedzę o tym co wydarzyło się podczas wykorzystywania zasobów, jak i pracy dla poszczególnych zespołów. Pozwala mu to na dokonanie wyboru zespołu, z którym chciałby współpracować w przyszłości.

12.4. Reprezentowanie zasobów w systemie

W proponowanym systemie dane dotyczące zasobów reprezentowane są w postaci instancji ontologii zasobów. Zauważmy tutaj, że dla proponowanego podejścia idealne byłoby istnienie powszechnie zaakceptowanej ontologii Gridu. Niestety, mimo istnienia wielu różnych ontologii opisujących Grid, są one niekompletne, niekompatybilne a ponadto często nie są ontologiami Gridu jako takiego, lecz ontologiami związanymi z Internetem semantycznym (Semantic Web). Dlatego zamiast wybierać jedną z nich, skoncentrowaliśmy się na aspektach agentowych proponowanego systemu, wykorzystując równocześnie uproszczone ontologie. Równocześnie należy podkreślić, że gdy powstanie powszechnie akceptowana ontologia opisująca Grid, proponowany tutaj system będzie przygotowany na jej wykorzystanie. Obecnie nasza ontologia koncentruje się na opisie aspektów obliczeniowych zasobów takich, jak procesor, pamięć, przestrzeń dyskowa. Poniżej przedstawiamy fragment (w notacji N3) ontologii (w języku OWL Lite) [11]:

```
:Computer
  :a owl:Class.
:hasCPU
  :a owl:ObjectProperty;
  rdfs:range :CPU;
  rdfs:domain :Computer.
:CPU
```

```
:hasMemory
  a owl:DatatypeProperty;
  rdfs:comment "in MB";
  rdfs:range xsd:float;
  rdfs:domain :Computer.
:hasUserDiskQuota
  :a owl:DatatypeProperty;
```

```
  :a owl:Class.

:hasCPUFrequency
  :a owl:DataProperty;
  rdfs:comment "in GHz";
  rdfs:range xsd:float;
  rdfs:domain :CPU.
:hasCPUType
  :a owl:ObjectProperty;
  rdfs:range :CPUType;
  rdfs:domain :CPU.
:CPUType
  :a owl:Class.
Intel :a :CPUType.
AMDAthlon :a :CPUType.
```

```
  rdfs:comment "in MB";
  rdfs:range xsd:float;
  rdfs:domain :Computer.
:LMaster
  :a owl:Class;
:hasContactAID
  :a owl:ObjectProperty;
  rdfs:range xsd:string;
  rdfs:domain :LMaster.
:hasUserDiskQuota
  :a owl:DatatypeProperty;
  rdfs:comment "in MB";
  rdfs:range xsd:float;
  rdfs:domain :Computer.
```

Dane opisanie ontologicznie przechowywane są (przez CIC) w repozytorium Jena 2.3 [8]. Dalej przedstawiamy instancję ontologii opisującej komputer z procesorem Intela o częstotliwości taktowania 3.0 GHz, który oferuje jako zasób gridowy 6 GB przestrzeni dyskowej, oraz 1024 MB pamięci.

```
:LMaster3
  :hasContactAID
"monster@e-plant:1099/JADE";
  :hasWorker :PC2929.

:PC2929
  :a :Computer;
  :hasCPU
  [
    a :CPU;
      :hasCPUType :Intel;
      :hasCPUFrequency "3.0"^^xsd:float;
  ] ;
  :hasUserDiskQuota "6000"^^xsd:integer;
  :hasMemory "1024"^^xsd:integer.
```

Podobnie do systemu e-commerce przedstawionego w rozdziale 11, w celu odpytania o potrzebne dane, zastosowaliśmy język SPARQL [14]. Poniżej prezentujemy zapytanie o komputer z procesorem Intel o prędkości obliczeniowej minimum 3.2 GHz, dostępnej pamięci dyskowej 3.5 GB, oraz pamięci głównej 512 MB.

```
PREFIX : <http://www.ibspan.waw.pl/mgrid#>
SELECT ?contact
WHERE
{
  ?lmaster
    :hasContactAID ?contact;
    :a :LMaster;
    :hasWorker
      [
        :a :Computer;
        :hasCPU
          [ a :CPU;
            :hasCPUType :Intel;
            :hasCPUFrequency ?freq;
          ];
        :hasUserDiskQuota ?quota;
        :hasMemory ?mem;
      ].
  FILTER (?freq >= "3.2"^^xsd:float)
  FILTER (?quota >= "3500"^^xsd:integer)
  FILTER (?mem >= "512"^^xsd:integer)
}
```

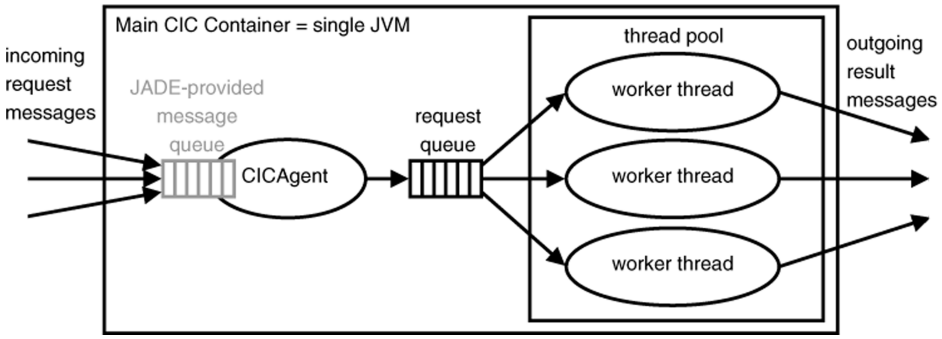
12.5. Efektywna implementacja centrum informacyjnego

Z przedstawionego opisu interakcji mających miejsce, gdy agenci reprezentujący użytkownika szukają zespołu posiadającego potrzebne im zasoby, lub do którego można się przyłączyć, oczywistym jest, że infrastruktura CIC jest jednym z kluczowych komponentów naszego systemu. Dlatego CIC powinien efektywnie obsługiwać duże liczby żądań od agentów. Ponieważ nasze rozwiązanie opiera się na scentralizowanym podejściu (*yellow pages*), postanowiliśmy dla niego znaleźć najbardziej optymalną implementację.

W [5] zaprezentowaliśmy kilka możliwych rozwiązań problemu implementacji infrastruktury CIC. W swoich poszukiwaniach bazowaliśmy na pracy [3], w której pokazaliśmy, że dla podobnego problemu najbardziej wydajnym było rozwiązanie opierające się na pojedynczym agencie, odbierającym i kolejkującym żądania użytkowników, oraz wielu agentach bazodanowych (SQLAgents), pobierających wiadomości z kolejki i wykonujących zapytania na bazie. Całe przetwarzania zapytania przez agentów, jak i sama baza danych zlokalizowane były na różnych komputerach. Przy pięciu SQLAgentach zaobserwowana wydajność wzrosła

o 33%. Dalej omawiamy dwie najlepsze architektury przedstawione w pracy [5]: (1) wielowątkową, (2) z rozproszonymi agentami bazodanowymi i dodatkowym wewnętrznym agentem CICIA.

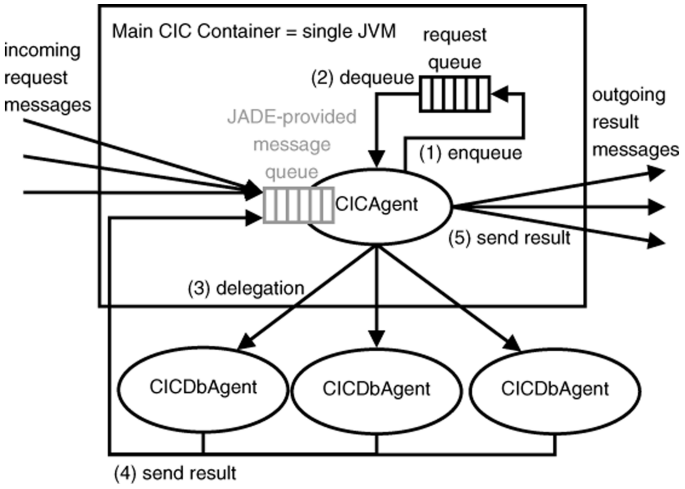
W architekturze wielowątkowej każde zadanie przetwarzane jest w osobnym wątku. Wykorzystujemy wątki Javy i udostępniamy je agentowi CIC w tym samym kontenerze. Każdy wątek ma swoje własne połączenie do bazy i swoją instancję modelu Jena. Ze względu na to, że tworzenie takich wątków jest czasochłonne, inicjujemy od razu pewną ich pulę. Agent CIC pobiera żądania (zapytania, bądź zlecenia aktualizacji danych) z kolejki wiadomości obsługiwanej przez JADE. Zauważmy tutaj, że każdy agent w platformie JADE posiada swoją (stworzoną przez JADE) kolejkę na wiadomości przychodzące od innych agentów. CIC odfiltrowuje te wiadomości, które dotyczą dostępu do bazy danych i kolejkuje je do wykonania przez pulę wątków. Po wykonaniu zapytania, odpowiedzi trafiają do agentów, którzy zlecili wykonanie zapytania (rys. 12.2).



Rysunek 12.2. Architektura wielowątkowa

Źródło: opracowanie własne.

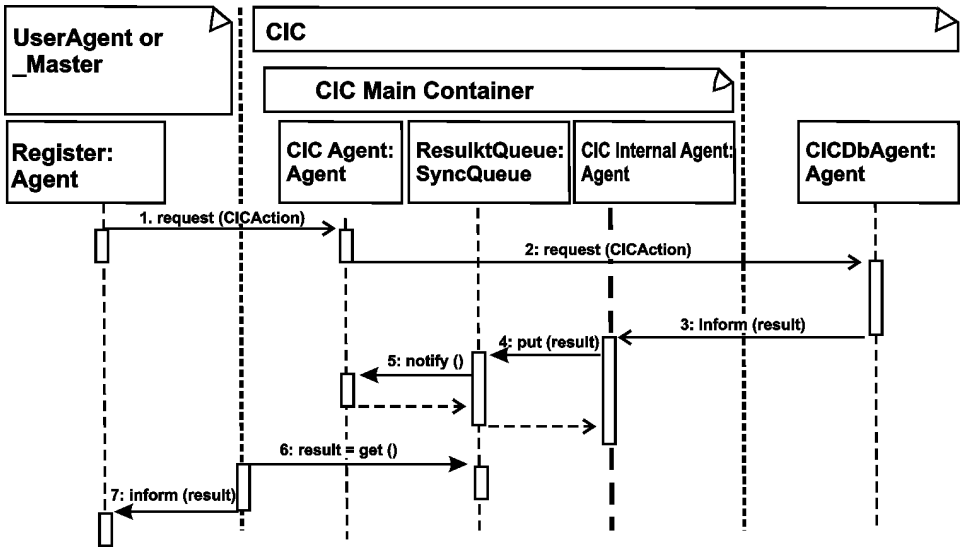
Kolejna testowana architektura opiera się na agentach CICDbAgent, zamiast na wątkach. Każdy z takich agentów uruchomiony jest na innym komputerze. Przychodzące żądania ponownie przechowywane są w kolejce JADE agenta CIC, z których zostają odfiltrowane wiadomości dotyczące zapytań bazodanowych, a następnie skolejkowane do wykonania przez agentów CICDbAgent. Agent CIC za pomocą metody „push" wypycha kolejne żądania z kolejki do wolnych agentów CICDbAgent. Każdy z agentów CICDbAgent wykonuje jedno zadanie na raz i odsyła wynik do wewnętrznego agenta CIC – CICIA (CIC Internal Agent), który z kolejki komunikatów JADE przemieszcza je do kolejki odpowiedzi (którą dzieli z agentem CIC). Z tej kolejki Agent CIC odsyła odpowiedzi do agentów, którzy nadesłali zapytania. Ze względu na współdzieloną przez CIC i CICIA kolejkę, obaj agenci muszą istnieć w tej samej maszynie wirtualnej, czyli w naszym przypadku w tym samym kontenerze (rys. 12.3).



Rysunek 12.3. Architektura z wieloma CICDbAgent i wewnętrznym agentem CICIA

Źródło: opracowanie własne.

W tej architekturze Agent CIC posiada trzy funkcjonalności: (1) odbiór żądań i kolejkowanie ich do przetworzenia, (2) wysyłanie żądań do agentów CICDbAgent, (3) odsyłanie odpowiedzi umieszczonych w kolejce przez CICIA. Na rysunku 12.4 przedstawiony jest diagram sekwencyjny interakcji LAgenta z agentem CIC w przypadku gdy mamy do czynienia z agentem CICIA jako pośrednikiem w przekazywaniu informacji.



Rysunek 12.4. Diagram sekwencji „rozmowy" Lagenta i CICAgenta

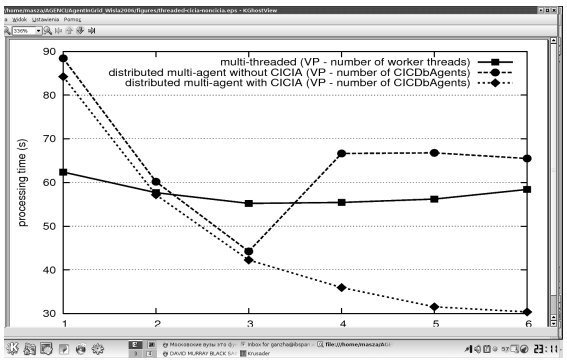
Źródło: opracowanie własne.

W naszych eksperymentach napływ żądań od agentów chcących wykonać zapytanie, został zasymulowany przez 4. agentów QA (Querying Agents), wysyłających do CIC zapytania w SPARQL. Zauważmy tutaj, że forma zapytania SPARQL może znacznie zmienić wydajność systemu. Silnik ARQ zastosowany w Jena jest odpowiedzialny za wykonywanie zapytań na ontologicznie opisanych zasobach przechowywanych w bazie danych. Tłumaczy on tylko część zapytania SPARQL na SQL. Pozostałą część (np. operacje FILTER), nie są wykonywane jako SQL, ale lokalnie przez ARQ, intensywnie wykorzystując zasoby JVM. W naszym przypadku zapytania miały następującą postać:

```
PREFIX grid : <http://gridagents.sourceforge.net/ Grid #>
SELECT ?master
WHERE
    ?comp grid:cpuClockSpeedMhz ?cpu .
    ?master grid:offersResource ?comp .
FILTER (? cpu > "1000"^^xsd:float)
```

Każdy agent QA uruchomiony był na oddzielnym komputerze i wysyłał 2500 żądań i odbierał odpowiedzi. Tak więc w przeprowadzonych eksperymentach generowano po 10.000 zapytań do przetworzenia przez CIC. Ze względu na konieczność przeprowadzenia wielu eksperymentów (konieczność dostosowywania parametrów pod kątem wydajności) stworzyliśmy testowy framework do automatycznego uruchamiania testów. Wszystkie eksperymenty koordynowane były przez TCA (Test Coordinator Agent). Przed każdym testem, zdalne kontenery JADE były restartowane w celu zapewnienia identycznych warunków testowych. Eksperymenty były przeprowadzane na 11 komputerach Athlon 2500+, 512MB RAM, z systemem operacyjnym Gentoo Linux i JVM 1.4.2. Komputery były połączone przez 100Mbit sieć LAN. MySQL w wersji 4.1.13 użyty do przechowywania danych przez Jena, uruchomiony był na osobnej maszynie. Procedura testu była następująca:

- restart zdalnych kontenerów,
- uczestniczący w eksperymencie agenci wysyłają do TCA wiadomość potwierdzającą gotowość,
- TCA po otrzymaniu sygnału gotowości od wszystkich, rozpoczyna test, wydając agentom QA polecenie start,
- kiedy agenci QA dostaną odpowiedzi na wszystkie swoje żądania, wysyłają wiadomość TCA o zakończeniu testu,
- odbiór wszystkich sygnałów o zakończeniu oznacza koniec eksperymentu.



Rysunek 12.5. Porównanie architektury wielowątkowej z architekturą CICDbAgent z i bez wewnętrznego agenta CICIA

Jak widać na wykresie, rozwiązanie oparte na wątkach, zyskuje na wydajności, gdy liczba wątków wzrasta do trzech. W przypadku wieloprocesorowej maszyny (lub wielordzeniowego procesora), wydajność mogłaby być większa, ale podejście to jest w każdym przypadku ograniczone przez całkowitą moc procesora/komputera.

W drugim rozwiązaniu liczba CICDbAgents zwiększona aż do sześciu dawała stały przyrost wydajności. Niestety nie mieliśmy możliwości przeprowadzenia eksperymentu na więcej niż 11. komputerach, stąd największa liczba wykorzystanych agentów CICDbAgent to 6. Można jednak przewidzieć (jest to już wyraźnie widoczne na rysunku), że dalsze zwiększanie liczby agentów CICDbAgent dałoby tylko niewielki przyrost. W porównaniu do pojedynczego agenta CICDbAgent, 6 agentów daje 3-krotne przyspieszenie. Rozwiązanie bazujące na wątkach w najlepszym wariancie (z trzema wątkami) dało 2-krotny przyrost wydajności.

Wykonaliśmy również pewną modyfikację drugiej architektury, w której agent CICIA stał się jednocześnie zleceniodawcą, jak i odbiorcą wyników. Okazała się ona bardziej wydajna, ale z drugiej strony również bardziej skomplikowana, i z tego względu została odrzucona.

12.6. Wybór zespołu, który wykona zadanie

Zajmijmy się teraz bardziej szczegółowym opisem operacji związanych z wybieraniem zespołu, który ma wykonać określone przez użytkownika zadanie. Pierwszym krokiem jest przekazanie wszystkich niezbędnych parametrów agentowi, takich jak: opis zadania, parametry negocjacyjne, dodatkowe ograniczenia.

12.6.1. Opis zadania i warunków jego wykonania

Użytkownik przekazuje LAgentowi opis zadania, parametry negocjacyjne i pewne ograniczenia. Parametry negocjacyjne przekazywane są z użyciem Grid Yellow Pages

Ontology (będącej drugą, poza ontologią zasobów, ontologią wykorzystywana w naszym systemie). Obecnie wykorzystujemy następujące parametry: koszt, czas rozpoczęcia obliczeń, czas zakończenia obliczeń. Każdy z parametrów posiada przypisane przez użytkownika wagi określające ważność [4, 13]. Oprócz parametrów przekazujemy również ich ograniczenia, tj. maksymalna cena. Jeśli któraś z wartości ma nie brać udziału w procesie negocjacji (jest dla nas nieistotna), przypisujemy jej wagę 0 lub pomijamy w zestawie parametrów. Możliwe jest również, że któryś z parametrów będzie posiadał ograniczenie, ale bez podanej wagi. Pokazuje to jak bardzo elastyczne i ekspresyjne jest podejście z wykorzystaniem ontologii. Poniżej prezentujemy schemat ontologii prezentujący opisane parametry i możliwość ich ograniczania.

```
### negotiation parameters ###
:NegotiationSet a owl:Class.
:hasNegotiationParam a owl:ObjectProperty;
        rdfs:domain :NegotiationSet;
        rdfs:range :NegotiationParam.
:NegotiationParam a owl:Class.
:paramWeight
        a owl:DatatypeProperty,
owl:FunctionalProperty;
        rdfs:domain: :NegotiationParam;
        rdfs:range xsd:float.
:Cost a owl:Class;
        rdfs:subClassOf :NegotiationParam.
:costConstraint
        a  owl:ObjectProperty,
owl:FunctionalProperty;
        rdfs:domain :Cost;
        rdfs:range :FloatConstraint.
:costValue
        a owl:DatatypeProperty,
owl:FunctionalProperty;
        rdfs:domain :Cost;
        rdfs:range xsd:float.
:JobStartTime a owl:Class;
        rdfs:subClassOf :NegotiationParam.
      :jobStartTimeValue
        a owl:DatatypeProperty,
owl:FunctionalProperty;
        rdfs:domain :JobStartTime;
        rdfs:range xsd:dateTime.
```

```
:jobStartTimeConstraint
        a owl:ObjectProperty, owl:FunctionalProperty;
        rdfs:domain :JobStartTime;
        rdfs:range :TimeConstraint.
### generic constraints ###
:NegotiationParamConstraint a owl:Class.
:FloatConstraint a owl:Class;
        rdfs:subClassOf :NegotiationParamConstraint.
:maxFloatValue
        a owl:FunctionalProperty,
owl:DatatypeProperty;
        rdfs:domain :FloatConstraint;
        rdfs:range xsd:float.
:minFloatValue
        a owl:DatatypeProperty,
owl:FunctionalProperty;
        rdfs:domain :FloatConstraint;
        rdfs:range xsd:float.
:TimeConstraint a owl:Class;
        rdfs:subClassOf :NegotiationParamConstraint.
:minDateValue
        a owl:DatatypeProperty,
owl:FunctionalProperty;
        rdfs:domain :TimeConstraint;
        rdfs:range xsd:dateTime.
:maxDateValue
        a owl:FunctionalProperty,
owl:DatatypeProperty;
        rdfs:domain :TimeConstraint;
        rdfs:range xsd:dateTime
```

Jak widać na schemacie oddzieliliśmy opis parametrów i ich ograniczeń na osobne klasy. Pozwala to nam na ponowne użycie ograniczeń do wielu parametrów. Przykładowo DateConstraint używany jest przez parametry JobStartTime i JobEndTime. Aktualnie ograniczenia pozwalają nam zdefiniować minimalne lub maksymalne wartości dla naszych parametrów, np. maksymalny koszt lub minimalny JobStartTime. Należy również zauważyć, że rozszerzenie zestawu parametrów, np. o karę za nie wykonanie zadania w określonym czasie, wymaga jedynie rozszerzenia naszej ontologii i dokonania drobnych zmian w kodzie agenta. Z tego względu skoncentrowaliśmy się na interakcji między agentami i wykorzystaniem parametrów i ograniczeń, a nie na stworzeniu pełnego zestawu możliwych parametrów. Załóżmy, że w przypadku gdyby użytkownik ustalił, że koszt wykonania zadania jest dwa razy ważniejszy (waga 2) niż czas zakończenia zadania (waga 1), wówczas zestaw parametrów wyglądałby następująco:

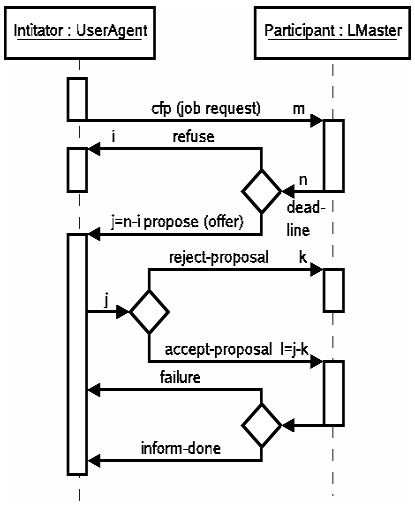
```
@prefix nego:
    <http://gridagents.sourceforge.net/Negotiation#>.

:NegotiationSetInstance a nego:NegotiationSet;
     nego:hasNegotiationParam [
       a nego:JobEndTime ;
       nego:paramWeight "1.0"^^xsd:float
     ], [
                a nego:Cost ;
       nego:paramWeight "2.0"^^xsd:float
     ].
```

Dodatkowo oprócz parametrów dotyczących wykonania zadania, użytkownik opisuje również zasoby wymagane do wykonania obliczeń (wykorzystując omówioną ontologię zasobów). Parametry te wykorzystywane są w zapytaniu SPARQL wykonywanym później przez agenta CIC.

12.6.2. Negocjacje

Do negocjacji z LMasterami LAgent wykorzystuje protokół FIPA Contract Net.

Na rysunku 12.6 przedstawiony jest schemat negocjacji użyty w naszym przypadku. Oczywiście negocjacje przeprowadzane są z wieloma LMasterami, a nie tylko z jednym. W pierwszym kroku LAgent wysyła wiadomość ACL o typie CFP (CALL-FOR-PROPOSAL) do wszystkich LMasterów na liście. Wiadomość CFP zawiera opis zadania i parametry zdefiniowane przez użytkownika. Oczywiście ograniczenia przypisane tym parametrom nie są wysyłane.



Rysunek 12.6. Contract Net Protocol

Źródło: opracowanie własne.

Przykład takiego zapytania znajduje się na następnej stronie:

```
(CFP :sender
  ( agent-identifier
    :name ua@kameleon:1099/JADE
    :addresses (sequence http://kameleon:7778/acc )
     ADE-agent-classname UserAgent )
  :receiver (set ( agent-identifier
    :name lmaster@e-plant:1099/JADE
    :addresses (sequence http://e-plant:7778/acc )) )
  :content „((action
    (agent-identifier :name lmaster@e-plant:1099/JADE
    :addresses (sequence http://e-plant:7778/acc))
    (JobRequest :resRequirements
    (OntoData
      :ontoDataLang RDF/XML-ABBREV
      :ontoDataStr
\"<rdf:RDF xmlns:grid="..." xmlns:rdf="...">
  <grid:UnitaryComputer
    rdf:about="jade://request@kameleon:1099/JADE">
  <rdf:type rdf:resource=
  "http://.../Grid#ComputerSystem"/>
  <grid:cpu>
    <grid:cpuClockSpeedMhz
```

```
        rdf:datatype="http://.../XMLSchema#int">
                    1500
      </grid:cpuClockSpeedMhz>
      </grid:cpu>
   </grid:UnitaryComputer>
</rdf:RDF>\"
      )
                    )
      :negoParamSet
       (NegotiationSet :hasNegotiationParam
       (sequence (JobEndTime
      :jobEndTimeConstraint (TimeConstraint
        :maxDateValue \"2006-10-12T12:00:00\")))
                    ))))"
  :reply-with  R1166131532630_0
  :language  fipa-sl0
  :ontology  Messaging
  :protocol  fipa-contract-net
  :conversation-id  C4916061_1166131532629 )
```

Opisuje ono zadanie, które wymaga komputera o procesorze z zegarem taktowania 1.5 GHz, oraz zakończenia zadania w czasie do 2006-10-12 o godzinie 12:00:00.

Każdy LMaster po otrzymaniu CFP przygotowuje swoją odpowiedź i odsyła w postaci wiadomości ACL o typie PROPOSE. Może również odpowiedzieć używając wiadomości REFUSE, odmawiając wykonania zadania. Na przykład, od momentu odpytania się LAgenta o zespoły, odmawiający wykonania zadania zespół stracił część swoich zasobów i nie może wykonać zadania. Poniżej przedstawiamy przykład wiadomości PROPOSE.

```
((PROPOSE
  :sender ( agent-identifier
    :name lmaster@e-plant:1099/JADE
    :addresses(sequence http://e-plant:7778/acc)
    :X-JADE-agent-classname LMaster )
  :receiver (set (agent-identifier
    :name ua@kameleon:1099/JADE
    :addresses(sequence http://kameleon:7778/acc)
    :X-JADE-agent-classname UserAgent ) )
  :content „(
    (result
      (action
```

```
            (agent-identifier
              :name lmaster@e-plant:1099/JADE
              :addresses(sequence http://e-plant:7778/acc)
              )
          (JobRequest
               ...
              )
           )
           (JobRequestOffer
            :negoParamSet
              (NegotiationSet :hasNegotiationParam
              (sequence
              (JobStartTime
                :jobStartTimeValue\"2006-10-12T11:30:00\")
              (JobEndTime
                 :jobEndTimeValue\"2006-10-12T12:30:00\")
               (Cost
                 :costValue \"120")
               )
              )
             )
         )
        )"
        :reply-with ua@kameleon:1099/JADE1166137976099
        :in-reply-to R1166137976093_0
        :language fipa-sl0
        :ontology Messaging
        :protocol fipa-contract-net
        :conversation-id C4916061_1166137976092
)
```

Wiadomość ta określa, że LMaster (i jego zespół) wykonają zadanie 2006-10-12 między godziną 10.15 a godziną 11.30; opłata to 120 jednostek umownych.

LAgent oczekuje na wiadomości PROPOSE przez określony czas (żeby uniknąć zakleszczeń, np. któryś LMaster nie odpowie bo stracił łączność). Jeśli minął timeout i LAgent nie dostał żadnej wiadomości PROPOSE, informuje o tym użytkownika i oczekuje na dalsze instrukcje. Jeśli nadejdzie przynajmniej jedna odpowiedź, LAgent przetwarza ją w dwóch krokach:

- oferty, które nie spełniają narzuconych ograniczeń (np. koszt, czas wykonania zadania) są odfiltrowywane na tym etapie; jeżeli żadna propozycja nie spełnia ograniczeń, LAgent informuje o tym użytkownika,
- pozostałe oferty są analizowane przez moduł analizy wielokryterialnej.

Dla wyjaśnienia dodajmy, że skoro agenci są autonomiczni to mogą przysłać oferty, które nie spełniają narzuconych ograniczeń. Tak więc oferty takie musimy odrzucić bez dalszego rozpatrywania.

Po wykonaniu pracy przez moduł analizy wielokryterialnej [4, 13] wyłoniony zostaje najlepszy zespół do wykonania zadania. Do tego zespołu wysyłana jest wiadomość ACCEPT-PROPOSAL. Wybrany zespół musi potwierdzić otrzymaną akceptację przez wiadomość INFORM-DONE (ponieważ LAgent był inicjatorem negocjacji Contract Net, więc zgodnie z protokołem to on musi otrzymać ostatnią wiadomość „domykającą protokół"). Wówczas pozostałe zespoły otrzymują REJECT-PROPOSAL. Oczywiście protokół Contract Net obsługuje również wyjątkowe sytuacje, np. brak odpowiedzi od zespołu.

12.7. Wybór zespołu do przyłączenia

Zajmijmy się teraz procesami mającymi miejsce, wtedy gdy agent poszukuje zespołu, do którego chciałby się przyłączyć. Proces ten inicjuje użytkownik, który dostarcza LAgentowi podstawowe informacje, tj. specyfikacja posiadanych zasobów, minimalne oczekiwane wynagrodzenie za godzinę pracy oraz dodatkowe ograniczenia, np. wyznaczone godziny pracy. Na podstawie tych danych LAgent stara się odnaleźć w Gridzie zespół, który odpowiada zdefiniowanym kryteriom. Tak, jak w przypadku zespołu szukającego zasobów, proces składa się z: (1) zapytania do CIC, (2) odfiltrowania otrzymanej listy zespołów, które nie są godne zaufania (np. nie wypłaciły należnej kwoty albo nie zrealizowały innych, ustalonych na etapie negocjacji, warunków) i (3) rozpoczęcia procesu negocjacji z przedstawicielami pozostałych zespołów. W trakcie tych negocjacji stosuje parametry przekazane przez użytkownika. Zespół, do którego przyłączy się LAgent, wybierany jest na podstawie analizy wielokryterialnej. Poniżej przedstawiamy szczegóły tego procesu.

12.7.1. Parametry definiowane przez użytkownika

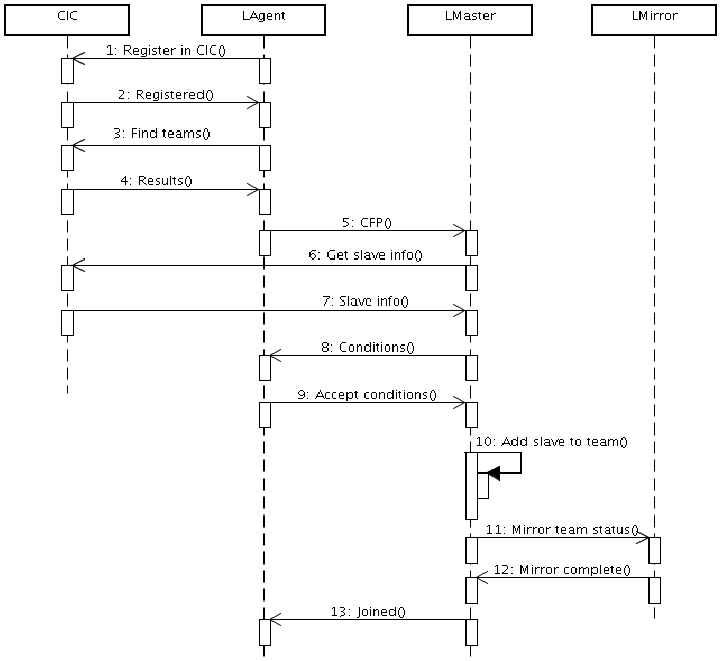
Obecnie używamy dwóch grup parametrów. Po pierwsze kryteria przyjęcia nowego Pracownika, definiowane przez zespół: wymagana liczba procesorów, częstotliwość taktowania, dostępna pamięć. Określają one minimalne wymagania dla nowych kandydatów. Druga grupa to wynagrodzenie i wymagany czas pracy. Określa ona warunki pracy członków zespołu. O ile na parametry należące do pierwszej grupy, nie mamy wpływu, gdyż opisują one posiadane przez nas zasoby, o tyle drugą grupę parametrów można dowolnie parametryzować. Możemy przyjąć, że pierwsza grupa to warunki, na podstawie których możemy zawęzić listę

zespołów w ramach zapytania do CIC. Natomiast druga stanowi parametry negocjacyjne. Każdy z parametrów negocjacyjnych ma przypisaną wagę. Jeśli nie bierzemy danego parametru pod uwagę, możemy ustawić wagę na 0. Dzięki temu, jeśli nasz komputer może pracować stale (24h), nie musimy uwzględniać parametru odpowiedzialnego za czas pracy, a jedynie za wynagrodzenie.

12.7.2. Negocjacje

Podobnie, jak w poprzednim przypadku, negocjacje o przystąpienie do zespołu wykorzystują protokół FIPA Contract Net. Na rysunku 12.7 przedstawiony jest diagram sekwencyjnego procesu.

Rysunek ten jest bardzo podobny do rysunku 12.6, zawiera jednak pewne dodatkowe elementy. W pierwszym kroku (wykonywanym tylko za pierwszym razem, gdy nowy LAgent zaczyna korzystać z systemu) LAgent rejestruje się w CIC. Ma to na celu zapewnienie, że tylko agenci zarejestrowani w systemie mają prawo w nim działać. Wprawdzie nie gwarantuje to zaufania do uczestników, ale stanowi mały krok w tym kierunku. Równocześnie LMaster po otrzymaniu CFP, (w kroku 6) kontaktuje się z CIC, w celu weryfikacji, czy dany LAgent jest zarejestrowany.



Rysunek 12.7. Diagram sekwencji negocjacji o przyjęciu agenta do zespołu

Źródło: opracowanie własne.

Jeżeli zasoby przedstawione w CFP spełniają wymagania LMastera, ten ostatni na podstawie swojej wiedzy o wykonanych pracach wykonuje obliczenia ceny za godzinę pracy agenta Pracownika w zespole. Ta cena zawiera w sobie ceny trzech podstawowych (bazowych) zasobów: M_b pamięci kosztuje c_M, P_b prędkości procesora kosztuje c_P i D_b dostępnej przestrzeni na dysku twardym kosztuje c_D. Jeżeli założymy, że zgłaszający się potencjalny Pracownik reprezentuje zasoby o następujących parametrach: pamięć M, prędkość procesora P i dostępną przestrzeń na dysku D, wówczas LMaster oblicza, że za jednostkę czasu może zapłacić temu Pracownikowi następną sumę jednostek umownych:

$$\text{Cost} = \alpha\left(\left(\frac{P}{P_b}\right)\times c_p + \left(\frac{M}{M_b}\right)\times c_M + \left(\frac{D}{D_b}\right)\times c_D\right)$$

gdzie α reprezentuje koszt własny LMastera.

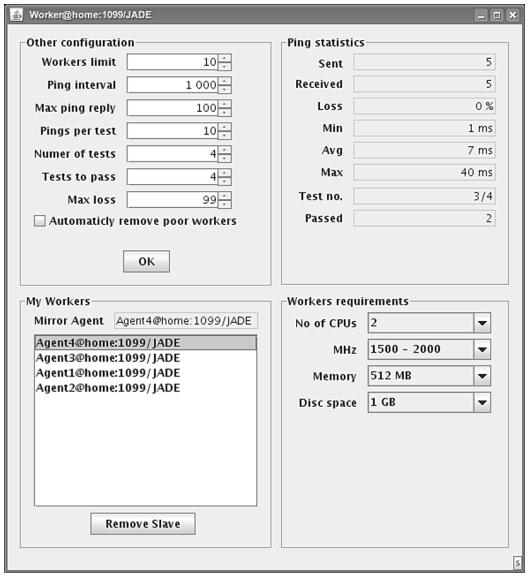
Pozostałe procesy mające miejsce w negocjacjach dotyczących przystąpienia do zespołu są praktycznie takie same, jak w przypadku wyboru zespołu do wykonania zadania i w związku z tym zostaną pominięte.

Zauważmy tutaj, że ponieważ w zespole istnieje LMirror, który odzwierciedla na bieżąco krytyczne do funkcjonowania LMastera informacje, musi on również zostać poinformowany o dołączeniu nowego Pracownika do zespołu. Gdyby LMaster nie replikował na bieżąco takich informacji, mogłoby dojść do sytuacji, w której LMaster dołączył nowego członka zespołu, po czym stracił łączność z resztą zespołu. Jego rolę przejąłby LMirror, który byłby nieświadomy istnienia nowego Pracownika. Prowadziłoby to do powstawania rozbieżności, których chcemy uniknąć.

W tym celu LMaster po otrzymaniu ACCEPT-PROPOSAL dodaje nowego członka zespołu do swoich lokalnych danych, po czym wymusza natychmiastową replikację na LMirrora. Dopiero w momencie otrzymania potwierdzenia od LMirrora, że dane zostały zreplikowane, do LAgenta wysyłane jest potwierdzenie za pomocą komunikatu INFORM-DONE. Gdyby dane nie zostały prawidłowo zreplikowane, LMaster uruchamia procedurę wyboru nowego LMirrora i próbuje dokonać replikacji. Gdyby zawiodły wszystkie próby replikacji, negocjacje z danym LAgentem zostają przerwane i zespół wchodzi w tryb awaryjny, który jest poza naszymi zainteresowaniami w tym rozdziale.

12.8. Implementacja

W chwili obecnej przedstawiony system jest w trakcie implementacji. Na rysunku 12.8 przedstawiamy konsolę LMastera.



Rysunek 12.8. Konsola LMastera

Źródło: opracowanie własne.

Na konsoli widzimy: (1) informacje o tym jakiego typu Pracowników poszukuje dany zespół (minimum 2 procesory, 1.5–2.0 GHz, 512 MB pamięci RAM i 1 GB pamięci dyskowej); (2) w chwili obecnej zespół LMastera składa się z 4. Pracowników (Agenci 1–4); (3) konfiguracje pakietu sprawdzania tego czy dany Pracownik jest nadal dostępny (określa ona, np. jak często sprawdzane będzie połączenie, jakie są zasady określania czy dany Pracownik jest uznawany za „żywego" lub „umarłego" etc.); oraz (4) statystyki ostatniej sesji sprawdzania żywotności Pracowników.

Uwagi końcowe

W tym rozdziale przedstawiliśmy agentowy system zarządzania zasobami w Gridzie. Wychodząc z założenia przedstawionego w pracy [6] i mówiącego, że agenci programowi mają do odegrania istotną rolę w Gridzie przeanalizowaliśmy istniejące wyniki badawcze i systemy dążące w tym kierunku. W wyniku przeprowadzonej analizy zaproponowaliśmy inne, bazujące na zespołach agentów podejście do tego zagadnienia. Przedstawiliśmy również najważniejsze cechy oraz funkcje systemu, który w chwili obecnej implementujemy.

Bibliografia

[1] Bădică C., Ganzha M., Gawinecki M., Kobzdej P., Paprzycki M.: *Towards Trust Management in an Agent-based E-commerce System – Initial Considerations*, In: ed. A. Zgrzywa, Proceedings of the MISSI 2006 Conference, Wrocław University of Technology Press, 2006, 225–236

[2] Cao J., Kerbyson D. J., Nudd G. R*., Use of Agent-Based Service Discovery for Resource Management in Metacomputing Environment* : Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, Springer-Verlag, London, UK, 2001, 882–886

[3] Chmiel K., Tomiak D., Gawinecki M., Kaczmarek P., Szymczak M., Paprzycki M., *Testing the Efficiency of JADE Agent Platform*, in: Proceedings of the ISPDC 2004 Conference, IEEE Computer Society Press, Los Alamitos, CA, 2004, 49–57

[4] Dodgson J., Spackman M., Pearman A., Phillips L., *DTLR multi-criteria analysis manual*, UK: National Economic Research Associates, 2001

[5] Dominiak M., Kuranowski W., Gawinecki M., Ganzha M., Paprzycki M.: *Efficient Matchmaking in an Agent-based Grid Resource Brokering System*, XXII Autumn Meetings of Polish Information Processing Society, November, PTI Press, 2006, 327–335

[6] Foster I., Jennings N. R., and Kesselman C.. *Brain meets brawn: Why grid and agents need each other*. Proceedings of the Autonomous Agents and Multi-Agent Systems, 2004, 8–16

[7] Foster I., Kesselman C., The Grid 2: *Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2003

[8] *Jena – A Semantic Framework for Java*, http://jena.sourceforge.net

[9] Manvi S.S., Birje, Prasad B., *An Agent-based Resource Allocation Model for computational grids*, Multiagent and Grid Systems, 1(1), 2005, 17–27

[10] Ouelhadj D., Garibaldi J., MacLaren J., Sakellariou R., Krishnakumar K., Amnon Meisels, *A Multi-agent Infrastructure and a Service Level agreement Negotiation Protocol for Robust Scheduling in Grid Computing*, LNCS, Advances in Grid Computing – EGC 2005, 3470/2005, 651–650

[11] OWL: www.w3.org/2004/OWL/

[12] Rana O. F., Di Martino B., *Grid performance and resource management using mobile agents, Performance analysis and grid computing*, 2004, 251–263

[13] Saaty T.L., *The Analytic Hierarchy Process* RWS Publications, Pittsburg, 1990

[14] SPARQL: www.w3.org/TR/rdf-sparql-query/

[15] Tianfield H., Unland R., *Towards self-organization in multi-agent systems and Grid computing, Multiagent and Grid Systems*, 1, 2005, 89–95

[16] Trastour D., Bartolini C., Preist C., *Semantic web support for the business-to-business e-commerce lifecycle* In: WWW '02: Proceedings of the 11th international conference on World Wide Web, 2002, 89–98