AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI
KATEDRA INFORMATYKI

Rozmieszczanie adaptacyjne aplikacji komponentowych w systemach rozproszonych

Streszczenie rozprawy doktorskiej

autor: mgr inż. Jacek Cała

promotor: prof. dr hab. inż. Krzysztof Zieliński

Czerwiec 2010

1 Wprowadzenie

W miarę postępującej ewolucji systemów komputerowych — od komputerów klasy *mainframe*, po dzisiejsze bardzo złożone systemy rozproszone — coraz większego znaczenia nabiera rozmieszczanie oprogramowania (ang. *software deployment*). Obecnie jest ono bardzo istotnym czynnikiem mającym wpływ na szereg aspektów działania systemów takich jak: wydajność, niezawodność czy bezpieczeństwo. Rozmieszczanie, jak to zostało trafnie wyrażone przez C. Szyperskiego,[1] jest elementem cyklu życia oprogramowania, którego zadaniem jest wypełnienie luki pomiędzy tym czego twórca oprogramowania nie może wiedzieć o środowisku wykonawczym, a tym czego administrator środowiska wykonawczego nie może wiedzieć o uruchamianym oprogramowaniu. Brak właściwego podejścia do rozmieszczania tworzy lukę, która niejednokrotnie prowadzi do problemów w działaniu oprogramowania.

Rozmieszczanie to proces, który oprogramowanie dostarczone przez producenta, udostępnia użytkownikowi poprzez: pobieranie, instalację i ew. uruchomienie, a w przypadku rozmieszczania pełnego umożliwia także aktualizację i rekonfigurację. Dobra infrastruktura rozmieszczania ułatwia zarządzanie oprogramowaniem poprzez automatyzację wielu niskopoziomowych zadań np. kopiowania elementów aplikacji czy koordynację uruchomienia systemu w środowisku rozproszonym. Ma ona także na celu zapewnienie, że elementy oprogramowania działają na możliwie najlepiej wybranych komputerach. Ponadto pełny model rozmieszczania rozszerza zakres rozmieszczania oprogramowania na fazę wykonania aplikacji przez co pozwala na reakcję na zmiany kontekstu, w którym działa oprogramowanie. Na przykład większe zainteresowanie użytkowników powoduje zwiększone obciążenie aplikacji co, jeśli zostanie wykryte, może zainicjować przeniesienie elementów oprogramowania na komputery o większych zasobach.

W przypadku otwartych rozproszonych systemów z heterogenicznymi zasobami proces rozmieszczania staje się zadaniem nietrywialnym. Dzieje się tak z powodu istnienia jawnych i niejawnych zależności pomiędzy oprogramowaniem i środowiskiem wykonawczym, a także różnorodności i zmiennej dostępności zasobów. Ponadto nowoczesne systemy często budowane są przy użyciu podejścia komponentowego. Poprzez podział aplikacji na wiele niewielkich, dobrze odseparowanych elementów ułatwia ono tworzenie i zarządzanie złożonymi systemami. Niestety większe rozdrobnienie elementów aplikacji związane z podejściem komponentowym jest także jedną z głównych przyczyn złożoności rozmieszczania.

W pracy pokazano, że dobrze określony model rozmieszczania oraz mechanizmy adaptacji pozwalają uprościć uruchamianie oprogramowania w otwartych środowiskach rozproszonych o heterogenicznych zasobach i mogą być podstawą do adaptacji aplikacji komponentowych. Efektem badań jest propozycja modelu rozmieszczania oraz jego prototypowa implementacja — platforma rozmieszczania adaptacyjnego ADF. W pracy przedstawiono projekt, implementację i ocenę jej działania.

[1] C. Szyperski, Przedmowa do materiałów konferencji Component Deployment 2002 w: *Proceedings of Component Deployment, IFIP/ACM Working Conference*, Berlin 2002.

1.1 Teza i cele badawcze rozprawy

W pracy sforumłowano następującą tezę:

> *Modern component-based systems can be successfully enhanced with a run-time reconfiguration mechanism and can enable deployment adaptation of component-based distributed systems,*

która w języku polskim brzmi:

> Nowoczesne systemy zbudowane w oparciu o podejście komponentowe mogą być z powodzeniem rozszerzone o mechanizmy rekonfiguracji w czasie działania umożliwiające adaptacyjne rozmieszczanie komponentów aplikacji rozproszonych.

Do wykazania tak postawionej tezy w pracy określono następujące cele badawcze:

- przeanalizowanie istniejących technik rozmieszczania, w tym rozmieszczania adaptacyjnego,
- opracowanie możliwie pełnego modelu rozmieszczania, który pozwalał będzie na adaptacyjne rozmieszczanie komponentów aplikacji w czasie wykonania,
- określenie i implementacja mechanizmów wymaganych do realizacji rozmieszczania w wymiarze przestrzennym, które stanowi podzbiór funkcjonalności opisanej przez opracowany model rozmieszczania,
- opracowanie i implementacja algorytmu planowania rozmieszczania dostosowanego do heterogenicznych środowisk rozproszonych oraz prowadzenia rekonfiguracji w czasie działania aplikacji,
- stworzenie prototypu platformy adaptacyjnego rozmieszczania komponentów, która łączy ww. elementny w spójną całość i pozwala na adaptację aplikacji według strategii określonej przez użytkownika,
- ocena efektywności stworzonego prototypu pod kątem zwiększenia wydajności działania aplikacji.

1.2 Osiągnięcia autora

Główne osiągnięcia badawcze przedstawione w pracy obejmują:

1. Opracowanie modelu rozmieszczania dla systemów komponentowych, który objemuje przestrzenny, czasowy i semantyczny wymiar planowania rozmieszczania i pozwala na modelowanie dynamicznych aspektów rozmieszczania takich jak aktualizacja i adaptacja.
2. Projekt, implementacja i ocena działania platformy adaptacyjnego rozmieszczania ADF, która umożliwia adaptację aplikacji rozproszonych w czasie działania.

3. Projekt, implementacja i ocena działania bazowych mechanizmów rozmieszczania adaptacyjnego takich jak: migracja komponentów w czasie działania, intercepcja komunikacji w warstwie aplikacji, monitorowanie infrastruktury.

4. Projekt i implementacja algorytmu FDDP — przybliżonego algorytmu planowania rozmieszczania klasy *force-directed algorithms*. Proponowany algorytm stanowi nowatorskie podejście do planowania rozmieszczania w czasie działania aplikacji.

2 Podstawy teoretyczne i prace związane z obszarem badań

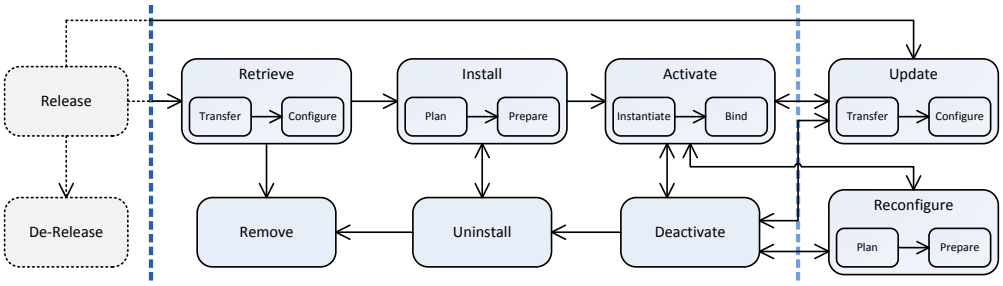
W podstawowej formie, rozmieszczanie oprogramowania definiowane jest jako proces zapoczątkowany pobranianiem oprogramowania od producenta i prowadzący do jego uruchomienia w środowisku konsumenta. Bardziej szczegółowa definicja rozmieszczania nie jest jednak jednoznacznie określona w literaturze. Przykładowo, Carzaniga i in. w [4] rozszerza pojęcie rozmieszczania o procesy związane z aktualizacją i adaptacją oprogramowania. W pracy zaproponowano rozróżnienie na dwie postaci rozmieszczania oprogramowania: prostą (ang. *plain deployment*) nazywaną też krótko rozmieszczaniem oraz adaptacyjną (ang. *adaptive deployment*), która bliższa jest propozycji Carzanigi, a którą zdefiniowano następująco:

> Rozmieszczanie adaptacyjne to ciągły proces wykonywany w środowisku konsumenta oprogramowania, który rozpoczyna się po opublikowaniu oprogramowania przez producenta i prowadzi do wykonania aplikacji. Podczas wykonania rozmieszczanie adaptacyjne dostarcza mechanizmów na potrzeby aktualizacji i rekonfiguracji oprogramowania.

Ilustracją do powyższej definicji może być diagram przedstawiony na rysunku 1, na którym aktywności zostały podzielone na trzy grupy. Najbardziej na lewo znajdują się akcje podejmowane przez producenta oprogramowania i, choć nie należą one do samego procesu rozmieszczania, stanowią dla niego kontekst. W części środkowej pokazano aktywności tworzące proces rozmieszczania prostego. Najbardziej na prawo zaprezentowane są dwie aktywności uzupełniające proste rozmieszczanie o aktualizację i rekonfigurację co tworzy kompletny proces rozmieszczania adaptacyjnego.

2.1 Automatyzacja rozmieszczania

Automatyzacja rozmieszczania stanowi podstawę dla rozmieszczania adaptacyjnego. Im bardziej zautomatyzowany jest proces rozmieszczania tym płynniej może przebiegać adaptacja procesu rozmieszczania, a pełna automatyzacja wszystkich jego aktywności prowadzi do rozmieszczania autonomicznego. Zaletą wprowadzenia



Rysunek 1: Diagram aktywności przedstawiający proces adaptacyjnego rozmieszczania oprogramowania

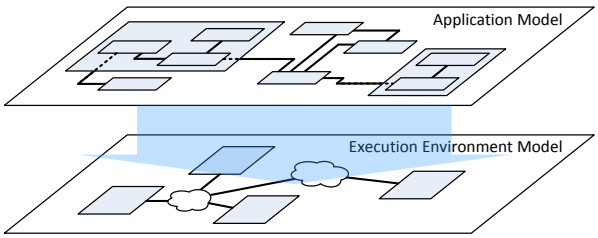
automatyzacji rozmieszczania jest ponadto udokumentowanie sposobu w jaki należy uruchamiać oprogramowanie. Mając taką dokumentację upraszcza się problem wielokrotnego uruchomienia oprogramowania.

Obecnie poziom automatyzacji rozmieszczania oprogramowania jest bardzo zróżnicowany i w dużej mierze zależy od złożoności środowiska wykonawczego. Dla środowisk najprostszych, złożonych z pojedynczego komputera, najtrudniejsze elementy rozmieszczania tj. planowanie i rekonfiguracja są wyeliminowane. Stąd też istnieje bardzo wiele rozwiązań i narzędzi, które oferują pełną automatyzację rozmieszczania. Najbardziej znaczące są tu narzędzia wykorzystujące sieć Internet m.in. Java Web Start [21], .Net ClickOnce[2] czy ZeroInstall.[3]

Inaczej jest w przypadku systemów rozproszonych. Rozmieszczanie oprogramowania w środowisku rozproszonym składającym się z N komputerów jest bardziej niż N-krotnie złożone. Jednym z czynników, które znacząco zwiększają trudność rozmieszczania jest heterogeniczność zasobów. Wymusza ona konieczność dokładnej specyfikacji wymagań oprogramowania oraz możliwości zasobów i prowadzi do trudnego problemu planowania rozmieszczania. Innymi istotnymi barierami na drodze rozmieszczania w środowiskach rozproszonych są: koordynacja procesu pomiędzy wieloma komputerami, problemy z zależnościami pojawiające się w przypadku rekonfiguracji, a nawet, co zauważa Gokhale i in. w [11], spójna konfiguracja wszystkich elementów systemu rozproszonego.

Talwar i in. w [23] wyróżnia trzy podejścia do problemu automatycznego rozmieszczania w środowisku rozproszonym: podejście skryptowe (ang. *script-based*), oparte o specjalizowany język (ang. *language-based*) i oparte o modele (ang. *model-based*). Spośród nich, podejście oparte o modele jest najbardziej zaawansowane i zostało wykorzystane w pracy jako podstawa do automatyzacji rozmieszczania oprogramowania. Kluczowymi elementami tego sposobu rozmieszczania jest dostarczenie modelu opisu oprogramowania i modelu opisu środowiska wykonawczego, a następnie odwzorowanie pierwszego modelu w drugi (rys. 2). Ta wyraźna separacja pozwala z jednej strony na wielokrotne wykorzystanie modelu oprogramowania do

[2] `http://msdn.microsoft.com`
[3] `http://0install.net`



Rysunek 2: Rozmieszczanie w podejściu opartym o modele separuje model oprogramowania od modelu środowiska wykonawczego

rozmieszczania w różnych środowiskach wykonawczych, a z drugiej na wielokrotne wykorzystanie modelu środowiska do rozmieszczania różnych aplikacji. Ponadto, w przypadku aplikacji komponentowych, model oprogramowania wynika wprost z architektury aplikacji co ułatwia wykorzystanie tego podejścia.

W pracy omówiono krótko dwie specyfikacje dotyczące rozmieszczania opartego o modele: *Common Information Model* [8] oraz *Deployment and Configuration of Component-based Applications* [18]. Druga z nich jest uważana za jedną z najbardziej kompletnych definicji rozmieszczania i konfiguracji [7], dlatego też stała się podstawą do dalszych badań w kierunku rozmieszczania adaptacyjnego.

2.2 Planowanie rozmieszczania

Rozważając rozmieszczanie oprogramowania w podejściu opartym o modele należy zwrócić uwagę na fazę planowania rozmieszczenia. W pracy przedstawiono definicję planowania oraz wyróżniono i zdefiniowano jego trzy wymiary: czasowy, przestrzenny i semantyczny. Problem rozmieszczania oprogramowania może być rozpatrywany w jednym z ww. wymiarów lub w dowolnej ich kombinacji przez co możliwe jest modelowanie wyszukanych scenariuszy rozmieszczania.

Do efektywnej realizacji rozmieszczania adaptacyjnego konieczna jest automatyzacja fazy planowania. Niestety problem planowania rozmieszczenia oprogramowania w środowiskach rozproszonych o heterogenicznych zasobach jest zadaniem NP-trudnym. Ponadto otwartość środowiska wykonawczego powoduje, że nieprzewidywalna staje się dostępność zasobów. Trudność planowania wynika zatem nie tylko ze złożoności obliczeniowej samego problemu, ale także: (1) ze zmian w dostępności zasobów środowiska, które są wynikiem współdzielenia węzłów obliczeniowych i sieci komputerowej oraz (2) ze zmian w wymaganiach aplikacji, które często zależą od czynników zewnętrznych. W tak dynamicznym środowisku planowanie, które uwzględnia tylko statyczny opis wymagań oprogramowania i dostępności zasobów, jest niewystarczające. Stąd też konieczność wprowadzenia do procesu rozmieszczania adaptacji.

2.3 Rozmieszczanie adaptacyjne

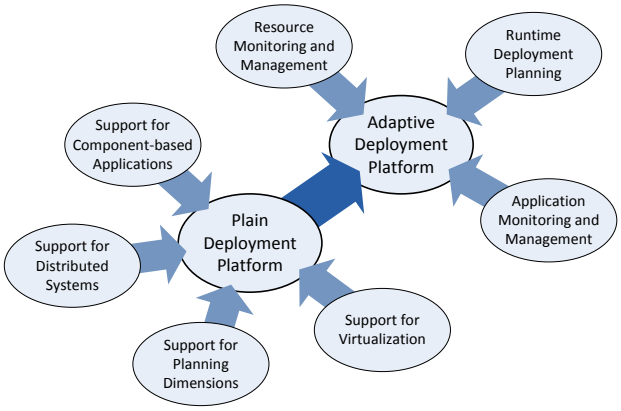
Rozpoznanie literatury oraz badania wstępne [1] pozwoliły ustalić, że wprowadzenie adaptacji do procesu rozmieszczania ma korzystny wpływ na wiele aspektów związanych z uruchomieniem i działaniem aplikacji. Kluczowa jest przy tym możliwość reakcji na zmiany w kontekście wykonania, a przez to możliwość dynamicznej rekonfiguracji oprogramowania w czasie działania. To pozwala rozwiązać problem planowania rozmieszczenia w czasie wykonania poprzez zastosowanie algorytmów przybliżonych i heurystyk, które biorą pod uwagę aktualny stan systemu. Ponadto upraszcza się opis zasobów i wymagań komponentów — może on zawierać wartości chwilowe zmiennych zamiast funkcji dostępności/wymagań. Kolejną zaletą zastosowania adaptacji w procesie rozmieszczania jest możliwość uniknięcia pesymistycznej rezerwacji zasobów. W przypadku, gdy dwa komponenty zaczynają wyczerpywać zasoby wspólnego węzła obliczeniowego system rozmieszczania adaptacyjnego może jeden z nich przenieść na inny, mniej obciążony węzeł.

Te i inne wymienione w pracy korzyści powodują, że problem adaptacyjnego rozmieszczania oprogramowania jest ważnym i ciekawym przedmiotem badań, a przegląd istniejących rozwiązań pozwala stwierdzić, że nie istnieje platforma rozmieszczania adaptacyjnego, która:

- operuje na aplikacjach zbudowanych w oparciu o zaawansowany model komponentowy,
- uwzględnia trzy wymiary planowania rozmieszczania,
- działa w otwartych środowiskach wykonawczych o heterogenicznych zasobach,
- w wysokim stopniu zapewnia automatyzację rozmieszczania,
- pozwala na rekonfigurację oprogramowania w czasie działania i
- o ile to możliwe korzysta z istniejących w tym zakresie standardów.

3 Koncepcja platformy rozmieszczania adaptacyjnego

Na zaprezentowaną w pracy koncepcję platformy rozmieszczania adaptacyjnego składają się dwa główne elementy: (1) infrastruktura prostego rozmieszczania oraz (2) infrastruktura rozmieszczania adaptacyjnego (rys. 3). Przy tworzeniu koncepcji zaczerpnięto z dwóch istotnych źródeł związanych z rozmieszczaniem aplikacji: specyfikacji *Deployment and Configuration of Component-based Applications* (D&C) [18] oraz modelu *Common Information Model* (CIM) [8]. D&C szczegółowo odnosi się do rozmieszczania aplikacji komponentowych w środowiskach rozproszonych, opisuje m.in. czym jest komponent, jego konfiguracja, czym jest środowisko wykonawcze. CIM natomiast definiuje bardzo szczegółowy model zasobów komputerowych, przez co pozwala zmniejszyć problem koordynacji opisu zasobów zauważony przez E. Kotsovinosa w [13].



Rysunek 3: Kluczowe elementy stanowiące podstawę koncepcji platformy adaptacyjnego rozmieszczania oprogramowania

Przy projektowaniu infrastruktury prostego rozmieszczania przyjęto następujące założenia:

- wykorzystanie środowiska budowy aplikacji komponentowych możliwie zgodnego z definicją komponentu proponowaną przez C. Szyperskiego w [22] oraz specyfikacją D&C. Istotne w tej definicji są stwierdzenia, że komponent to jednostka budowy aplikacji, enkapsulacji stanu oraz niezależnego rozmieszczania, która wyraża zależności względem otoczenia poprzez wymagane i dostarczane interfejsy. Zależności względem otoczenia mogą odnosić się do innych komponentów, ale też do kontekstu wykonania. Pozwala to na separację komponentu względem środowiska i jest istotne przy budowie mechanizmów rekonfiguracji,

- wykorzystanie podejścia do rozmieszczania opartego o modele, które pozwala na pełną automatyzację fazy planowania i może być podstawą dla rozmieszczania adaptacyjnego. Ponadto rozmieszczanie w środowisku rozproszonym podzielono na dwa poziomy abstrakcji: globalny i lokalny. Na poziomie globalnym dotyczy ono całości aplikacji i środowiska wykonawczego, a na poziomie lokalnym koncentruje się na pojedynczym węźle wykonawczym i przypisanej do niego grupie komponentów,

- uwzględnienie wszystkich trzech zdefiniowanych wcześniej wymiarów planowania. W wymiarze przestrzennym infrastruktura powinna umożliwiać opis wymagań komponentów, zasobów środowiska i przypisanie komponentów do węzłów środowiska. W wymiarze semantycznym powinna dawać możliwość opisu wymagań niefunkcjonalnych komponentów, zasobów dostępnych usług i przypisania komponentów do usług. W wymiarze czasowym powinna pozwalać na określenie zależności czasowych pomiędzy komponentami wraz z ich wymaganiami co do zasobów, a także powinna umożliwiać określenie porządku rozmieszczania komponentów,

- nie uwzględnienie wirtualizacji środowiska wykonawczego w rozmieszczaniu. Pomimo, że wirtualizacja jest obecnie bardzo powszechna i stanowi istotny aspekt w rozwiązaniu problemu rozmieszczania, opracowanie modelu, który dawałby możliwość rozmieszczania w środowisku zwirtualizowanym jest zadaniem wymagającym odrębnych badań. W pracy ograniczono się do przedstawienia najistotniejszych czynników, które powinny być wzięte pod uwagę przy budowie takiej platformy, a które są wynikiem prowadzonych wcześniej prac [3].

Na bazie infrastruktury prostego rozmieszczania opracowano koncepcję infrastruktury adaptacji. Podstawowe do stworzenia platformy rozmieszczania adaptacyjnego wydało się być zastosowanie technik refleksji strukturalnej (ang. *architectural reflection*) [16, 6] i przetwarzania autonomicznego (ang. *autonomic computing* — AC) [12].

Refleksja strukturalna zakłada odseparowanie aplikacji i środowiska wykonawczego od ich samoreprezentacji. Celem tej separacji jest adaptacja systemu w czasie działania poprzez synchronizację stanu samoreprezentacji z reprezentowanym obiektem. Konieczne jest przy tym dostarczenie mechanizmów obserwacji i manipulacji, które taką synchronizację umożliwią. Infrastruktura prostego rozmieszczania zbudowana zgodnie z podejściem opartym o modele daje naturalną możliwość stworzenia samoreprezentacji zarówno aplikacji jak i środowiska wykonawczego. Rozwiązanie w pełni reflektywne uzyskać można zatem dostarczając odpowiednie mechanizmy obserwacji i manipulacji.

Przetwarzanie autonomiczne zakłada natomiast separację architektury systemu na warstwę logiki adaptacji oraz warstwę monitorowania i zarządzania zasobami. W efekcie ułatwia ono modyfikację i rozbudowę systemów oraz daje możliwość tworzenia hierarchii elementów zarządzających.

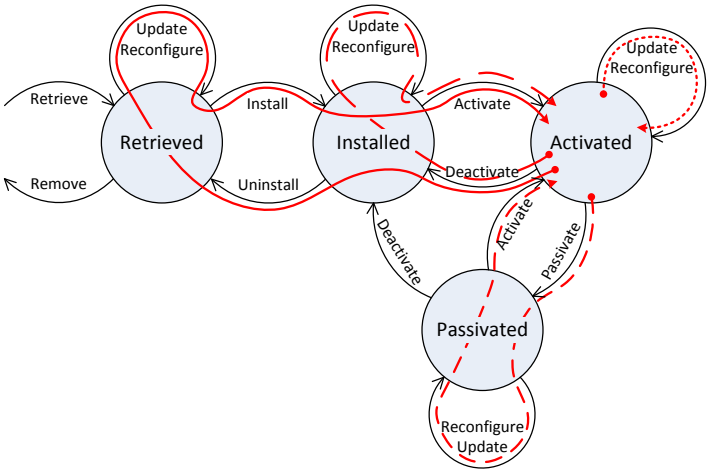
Budowa infrastruktury rozmieszczania adaptacyjnego wymaga więc opracowania:

- monitorowania środowiska wykonawczego,
- monitorowania aplikacji w czasie działania,
- mechanizmów ponownego rozmieszczania (ang. *redeployment*), które umożliwią rekonfigurację aplikacji,

a w warstwie logiki adaptacji:

- algorytmu adaptacji kontrolującego proces rozmieszczania zgodnie z celem adaptacji aplikacji.

Do monitorowania zasobów środowiska rozproszonego użyto powszechnie zaakceptowanego standardu CIM. Dostępność implementacji infrastruktury CIM (WBEM) na różnych platformach sprzętowych i programowych powoduje, że może on być także wspólnym językiem opisu zasobów. Dużo trudniej o rozwiązanie problemu monitorowania aplikacji. W tym zakresie CIM ograniczony jest tylko do monitorowania systemów klasy *Java Enterprise Edition*. Koncepcja platformy zakłada zatem własne rozwiązanie opisu zmiennych obserwowanych w warstwie aplikacji.



Rysunek 4: Diagram stanów procesu rozmieszczania adaptacyjnego z wyróżnionymi czterema technikami ponownego rozmieszczania: pełne (linia ciągła); głębokie (linia przerywana — —); płytkie (linia przerywana – –); w czasie wykonania (linia kropkowana)

Istotnym elementem infrastruktury adaptacyjnego rozmieszczania jest dostępność mechanizmów ponownego rozmieszczenia po uruchomieniu aplikacji (ang. *redeployment*). W pracy zdefiniowano cztery możliwe techniki ponownego rozmieszczania: pełne, głębokie, płytkie oraz ponowne rozmieszczanie w czasie wykonania. Na rysunku 4 przedstawiono diagram stanów rozmieszczania adaptacyjnego, na którym wyróżniono ścieżki reprezentujące ww. techniki. Do ich realizacji infrastruktura rozmieszczania powinna dostarczać niskopoziomowe mechanizmy zarządzania komponentami takie jak: zachowanie stanu, wstrzymanie, przełączanie, migrację w czasie działania, wirtualne rozmieszczanie.

W pracy rozważono zastosowanie w adaptacji wszystkich wymienionych technik. Ostatecznie jednak proponowana platforma realizuje najbardziej zaawansowane — ponowne rozmieszczanie w czasie wykonania. Pozwala ono na zmianę rozmieszczenia komponentów „w locie” i jest najmniej inwazyjną metodą rekonfiguracji.

Mając do dyspozycji mechanizmy monitorowania i manipulacji uzupełnienia wymaga ponadto warstwa logiki adaptacji. Z uwagi na złożoność zagadnienia, w pracy poświęcono temu osobny rozdział. Proponowane rozwiązanie wykorzystuje algorytm klasy *Force-Directed Algorithms* i omówione zostało w punkcie 7.

4 Platforma rozmieszczania adaptacyjnego

Przedstawiona w pracy prototypowa platforma rozmieszczania adaptacyjnego implementuje tylko wybrane elementy całego modelu. Opracowanie pełnej implementacji modelu rozmieszczania, który obejmuje trzy wymiary planowania i pozwala na

adaptację z użyciem dowolnej z czterech technik ponownego rozmieszczania, jest wyzwaniem znacząco wykraczającym poza ramy tej pracy. Dlatego też, platforma implementuje jedynie te aspekty modelu, które pokazują, że może on zostać choć częściowo zrealizowany i może być wykorzystany do adaptacji aplikacji. Prace implementacyjne podzielono na następujące trzy fazy:

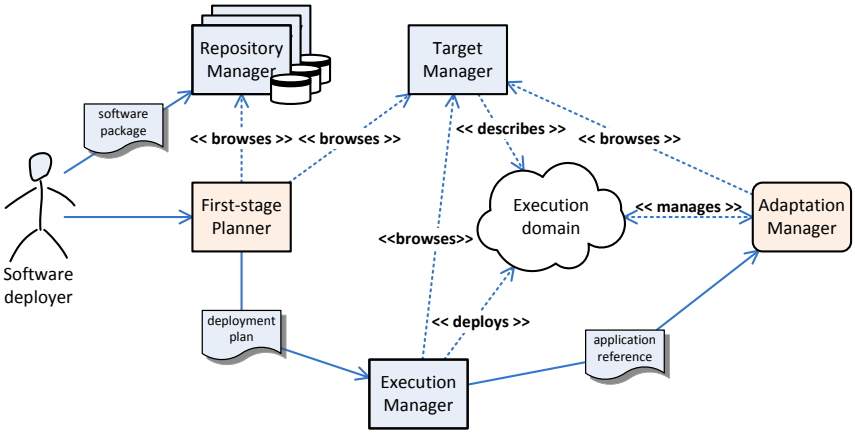
- stworzenie infrastruktury prostego rozmieszczania zawężonego do wymiaru przestrzennego,
- opracowanie algorytmów planowania i stworzenie komponentu planisty; w tym przypadku planista realizuje wyłącznie statyczne planowanie początkowe tj. przed uruchomieniem aplikacji,
- implementacja infrastruktury adaptacji, w skład której wchodzą mechanizmy monitorowania, rekonfiguracji i planowania w czasie wykonania.

Istotnym założeniem, które przyjęto w czasie prowadzonych badań jest wykorzystanie modelu CORBA Component Model (CCM) jako technologii tworzenia komponentowych aplikacji dla środowisk rozproszonych. Model CCM zaproponowany przez OMG w [20], choć nie zyskał popularności, posiada bardzo wiele wartościowych cech, które decydują o jego przydatności w tworzeniu systemów rozproszonych. Realizuje on wiele istotnych wzorców projektowych m.in.: *dependency injection*, późne wiązanie, dwu-fazową inicjalizację, programowanie zdarzeniowe, separację pomiędzy komponentem i kontekstem wykonania. Połączenie modelu CCM wraz z uszczegółowieniem modelu rozmieszczania (PSM for CCM) opracowanym w specyfikacji D&C stanowi, według autora, jedną z najbardziej zaawansowanych technologii tworzenia oprogramowania dla heterogenicznych środowisk rozproszonych.

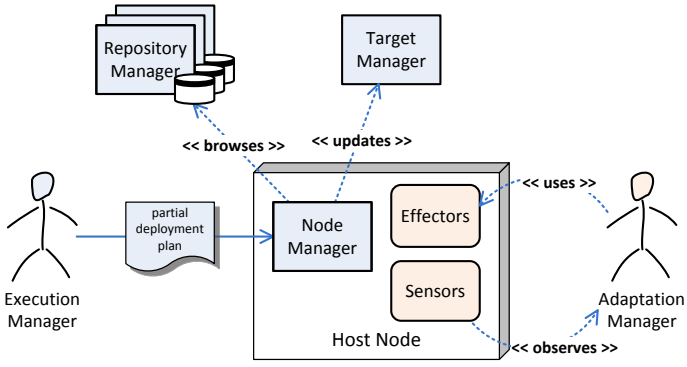
4.1 Zarys architektury platformy ADF

Ogólna architektura stworzonej platformy ADF przedstawiona została na rysunkach 5 i 6. Pokazują one odpowiednio widok globalny i lokalny platformy oraz różnice w stosunku do architektury zaproponowanej w specyfikacji D&C. Na widoku globalnym wyróżniono dwa elementy, które rozszerzają model D&C: komponent planisty początkowego (*First-stage Planner*) i komponent menadżera adaptacji (*Adaptation Manager*). Po zleceniu rozmieszczenia aplikacji przez aktora *Software Deployer* planista poszukuje początkowego planu rozmieszczenia. Pobiera w tym celu informacje o pakiecie oprogramowania od menadżera repozytorium (*Repository Manager*) a informacje o środowisku wykonawczym (*Execution Domain*) od menadżera środowiska docelowego (*Target Manager*). Opracowany przez planistę plan rozmieszczenia jest przekazywany do menadżera wykonania (*Execution Manager*). Ten, po uruchomieniu aplikacji, przekazuje jej referencję menadżerowi adaptacji. Zadaniem menadżera adaptacji jest obserwacja środowiska i wykonującej się w nim aplikacji oraz stosowna rekonfiguracja rozmieszczenia komponentów.

W widoku globalnym operacje rozmieszczania dotyczą całego środowiska wykonawczego. W widoku lokalnym przebiega ono na poziomie pojedynczego węzła



Rysunek 5: Globalny widok architektury platformy ADF; elementy wyróżnione pokazują rozszerzenia względem bazowej architektury D&C



Rysunek 6: Lokalny widok architektury platformy ADF; elementy wyróżnione pokazują rozszerzenia względem bazowej architektury D&C

(*Host Node*). Lokalnie akcja rozmieszczania rozpoczyna się od przekazania przez menadżera wykonania częściowego planu rozmieszania, który dotyczy danego węzła. Otrzymując taki plan menadżer węzła dokonuje uruchomienia odpowiednich komponentów aplikacji. W widoku lokalnym widoczny jest również menadżer adaptacji, który ma dostęp do węzła wykonawczego za pomocą sensorów i efektorów.

4.2 Infrastruktura prostego rozmieszczania

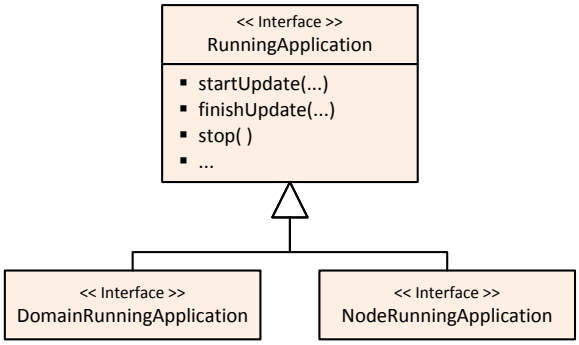
Funkcjonalność większości elementów infrastruktury prostego rozmieszczania jest zgodna z założeniami określonymi w specyfikacji D&C oraz w uszczegółowieniu modelu rozmieszczania dla CCM (PSM for CCM) [18, rozdz. 10] oraz [20, rozdz. 14]. Na uwagę zasługuje jednak komponent planisty początkowego, który bezpośrednio nie został ujęty w tych dokumentach. Pojawienie się planisty początkowego w architekturze platformy podyktowane zostało dwoma czynnikami:

- przyjęte środowisko wykonawcze jest otwartym systemem rozproszonym, w którym dostępność zasobów ulega ciągłym zmianom,
- zużycie zasobów przez komponenty jest zwykle zmienne i często zależy od zewnętrznych czynników, dlatego też jego odpowiednie modelowanie i ujęcie w statycznym opisie aplikacji jest trudne o ile w ogóle możliwe.

Z uwagi na ciągłe zmiany w środowisku wykonawczym planista powinien możliwie jak najszybciej dostarczyć plan rozmieszczenia oprogramowania. W przeciwnym razie plan może stać się nieefektywny, a w najgorszym przypadku niepoprawny (np. po awarii któregoś z węzłów docelowych). Ponieważ zagadnienie planowania rozmieszczenia jest problemem NP-trudnym, więc w środowiskach otwartych nie ma miejsca na dokładne, optymalne czy suboptymalne rozwiązania — planista musi generować plan szybko. Ponadto do dokładnego planowania rozmieszczenia komponentów w środowisku należałoby precyzyjnie określić wymagania komponentów co do zasobów. Niestety bardzo często zużycie zasobów przez komponent zależy od zewnętrznych czynników takich jak liczba użytkowników, czy rozmiar danych wejściowych. Samo opracowanie modelu zużycia zasobów przez komponent w zależności od czynników zewnętrznych jest zagadnieniem trudnym, a użycie takiego modelu w planowaniu tylko podnosi i tak wysoką złożoność obliczeniową problemu. Przyjęto więc, że planista początkowy operował będzie jedynie na statycznych wymaganiach komponentu określonych przez producenta i/lub użytkownika, a wygenerowany przez niego plan będzie poprawiany przez planistę czasu wykonania.

Podstawą działania planisty początkowego jest przeszukiwanie typu Best First Search (BFS) z heurystyką First-Fit (FF). W pracy omówiono sposób wykorzystania tego dobrze znanego algorytmu w odniesieniu do problemu planowania rozmieszczenia.[4] Przedstawiono tam również istotne aspekty algorytmu specyficzne dla modeli zdefiniowanych w specyfikacji D&C. Najważniejsze z nich to: sposób wyboru implementacji komponentu z grupy implementacji równorzędnych, problem dopasowania

[4]BFS jest jednym z podstawowych algorytmów dla rozwiązania problemu Bin Packing Problem [5].



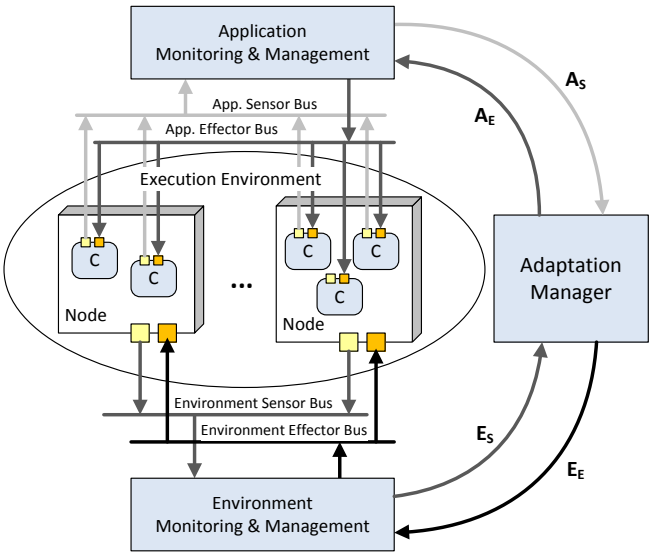
Rysunek 7: Interfejsy w infrastrukturze rozmieszczania prostego umożliwiające aktualizację rozmieszczenia aplikacji

wymagań połączeń do możliwości sieci, problem nałożenia ograniczeń przestrzennych. Dyskusja zawiera także krótką analizę złożoności obliczeniowej algorytmu.

Punktem styku infrastruktury prostego rozmieszczania z warstwą adaptacji są opracowane dodatkowe interfejsy, które umożliwiają dokonywanie aktualizacji i rekonfiguracji bieżącego rozmieszczenia (rys. 7). W pracy wyróżniono dwie metody aktualizacji rozmieszczenia: wewnętrzną i zewnętrzną. Aktualizacja wewnętrzna przeprowadzana jest całkowicie przez infrastrukturę rozmieszczania. Natomiast aktualizacja zewnętrzna powoduje, że infrastruktura rozmieszczania dokonuje tylko zmian w planie rozmieszczenia i wymaga zewnętrznego mechanizmu, który zsynchronizuje stan rozmieszczenia komponentów aplikacji z planem. Platforma ADF wykorzystuje oba sposoby aktualizacji. Wewnętrzny jest stosowany w przypadku aktualizacji w warstwie kontekstu wykonania komponentów (np. uruchomienie serwera czy kontenera dla komponentów), zewnętrzny — w przypadku aktualizacji w warstwie aplikacji. W opracowanej implementacji ADF mechanizmem uzupełniającym aktualizację zewnętrzną jest migracja komponentów w czasie wykonania.

4.3 Infrastruktura rozmieszczania adaptacyjnego

Infrastruktura rozmieszczania adaptacyjnego została zaprojektowana w oparciu o paradygmat przetwarzania autonomicznego (ang. *autonomic computing* — AC). Głównym jego założeniem jest separacja warstwy logiki adaptacji or warstwy monitorowania i zarządzania zasobami. Warstwy te są ze sobą połączone za pomocą jednolitego interfejsu sensorów i efektorów. W przypadku platformy ADF w warstwie monitorowania i zarządzania zasobami wyróżniono ponadto dwie podwarstwy: związaną ze środowiskiem wykonawczym i związaną z wykonywaną aplikacją. Na rysunku 8 przedstawiono model platformy ADF, na którym wyróżniono cztery możliwe pętle sterowania adaptacją. W prototypowej implementacji ADF wykorzystano dwie z nich: $A_S - A_E$ (monitorowanie i sterowanie w podwarstwie aplikacji) oraz $E_S - A_E$ (monitorowanie w podwarstwie środowiska wykonawczego i sterowanie w podwarstwie aplikacji). W ramach badań przeprowadzono również eksperymenty z pozostałymi



Rysunek 8: Model platformy ADF pokazujący separację na dwie podwarstwy monitorowania i zarządzania zasobami: podwarstwę aplikacji i podwarstwę środowiska wykonawczego. Stwarza to możliwość adaptacji z użyciem czterech pętli sterowania: $A_S - A_E$, $A_S - E_E$, $E_S - E_E$ oraz $E_S - A_E$

dwiema pętlami sterowania wykorzystując do tego celu mechanizm rozmieszczania wirtualnego [2].

Infrastruktura rozmieszczania adaptacyjnego została zbudowana w modelu komponentowym CCM. Przy użyciu tego modelu łatwo zrealizować podstawowe założenia AC. Pozwala on także na uruchomienie części adaptacyjnej ADF za pomocą opracowanej wcześniej infrastruktury prostego rozmieszczania.

Na potrzeby infrastruktury adaptacji zbudowano następujące komponenty:

- w podwarstwie monitorowania i zarządzania środowiskiem wykonawczym: sensory procesora i pamięci wykorzystujące model CIM,
- w podwarstwie monitorowania i zarządzania aplikacją: sensory wykorzystujące interceptory COPI, sensor komponentów (ang. *Instance Sensor*) oraz efektor migracji komponentów w czasie wykonania,
- w warstwie logiki adaptacji: menadżer adaptacji.

Poza wymienionymi komponentami, które w pracy przedstawiono bardziej szczegółowo, w ramach prac wstępnych stworzono również mechanizm rozmieszczania wirtualnego. Podstawą jego działania było wykorzystanie warstwy wirtualizacji systemowej do sterowania zasobami węzłów wirtualnych. Mechanizm ten mógłby stać się podstawą dla efektora domykającego pętle sterowania: $A_S - E_E$ i $E_S - E_E$. Niestety prace nad efektorem zostały wstrzymane, gdyż do właściwego działania

wymaga on stosunkowo dużego poziomu ziarnistości kontekstu wykonania komponentu. W przypadku znanych mechanizmów wirtualizacji systemowej najmniejszą jednostką zarządzania jest proces systemu operacyjnego.[5] Jednak pomiędzy procesem a komponentem CCM istnieje zwykle relacja 1-do-N. Powoduje to, że mechanizm rozmieszczania wirtualnego musiałby operować na grupie komponentów co nie zawsze jest pożądane, albo wymagałby uruchomienia każdego komponentu w osobnym procesie co wiąże się z dużym narzutem na wykonanie. Technika wirtualnego rozmieszczania może być interesującym polem dalszych badań, w przedstawionej pracy skupiono się jednak na wykorzystaniu bardziej elastycznego mechanizmu migracji komponentów w czasie wykonania.

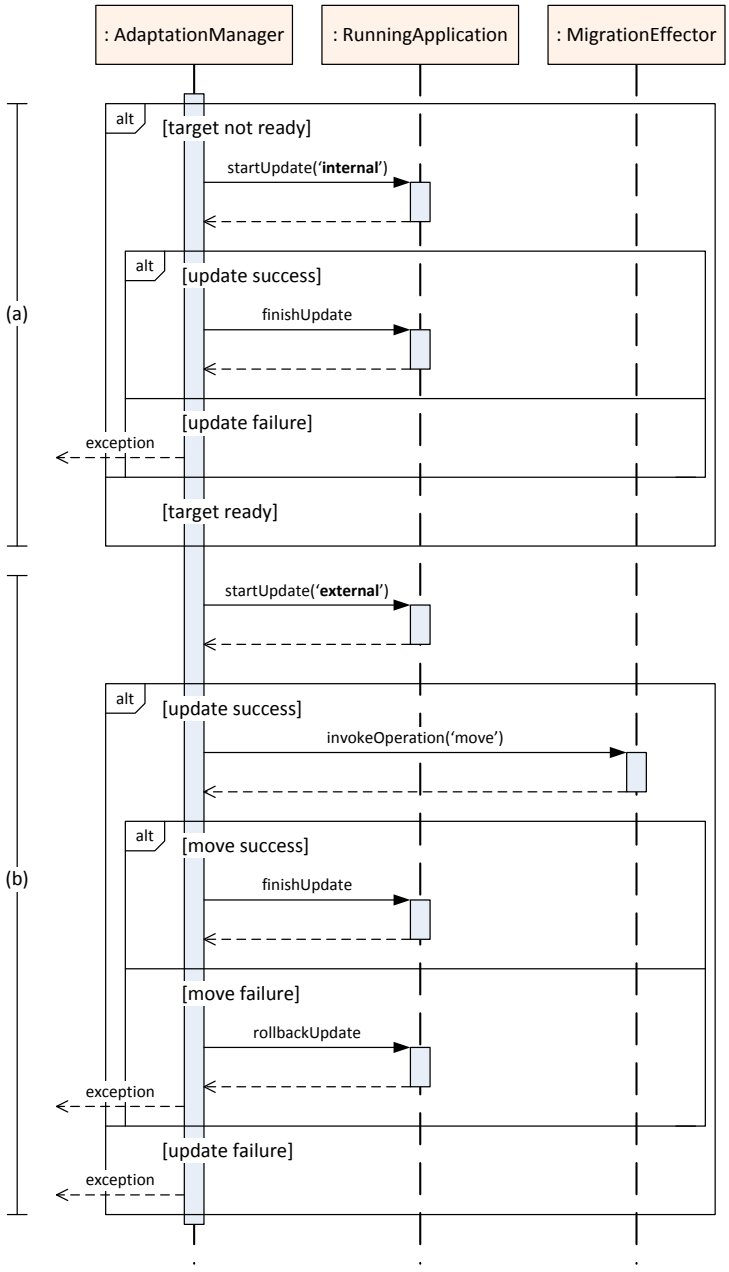
W warstwie logiki adaptacji skonstruowano komponent menadżera adaptacji. Istotnym założeniem przy jego projektowaniu była decyzja o tym, że menadżer odpowiedzialny jest za zarządzanie rozmieszczaniem komponentów pojedynczej aplikacji. Upraszcza to problem adaptacji, a przy tym w żaden sposób nie ogranicza rozwiązania. Zgodnie z podejściem AC menadżery adaptacji mogą być łączone w hierarchie. W tym wypadku menadżer adaptacji wyższego rzędu, mogłby koordynować pracę wielu menadżerów adaptacji poszczególnych aplikacji.

Zasadniczym problemem przy tworzeniu menadżera adaptacji była realizacja analizy i planowania, dwóch etapów pętli sterowania MAPE proponowanej przez podejście AC. Analiza w przypadku problemu rozmieszczania to zaplanowanie nowego, korzystniejszego rozmieszczenia komponentów na podstawie zebranych informacji o działaniu systemu. Natomiast etap planowania AC wiąże się z rekonfiguracją bieżącego rozmieszczenia zgodnie z nowym planem, który powstał w poprzednim kroku. Tu wykorzystano opracowany mechanizm migracji komponentów w czasie wykonania oraz metody wewnętrznej i zewnętrznej aktualizacji rozmieszczenia. Rysunek 9 przedstawia diagram sekwencji obrazujący interakcję pomiędzy elementami platformy ADF. Na pojedynczą iterację pętli adaptacji MAPE w etapie planowania wykonywane są dwa kroki: (a) weryfikacja i ew. przygotowanie kontekstu wykonania komponentów tj. uruchomienie potrzebnych serwerów komponentów i kontenerów, (b) rekonfiguracja aplikacji z wykorzystaniem aktualizacji zewnętrznej i efektora migracji.

5 Infrastruktura monitorowania i zarządzania

W podejściu AC infrastruktura monitorowania i zarządzania stanowi fundament dla działania autonomicznego menadżera. Jak wspomniano wyżej w przypadku platformy ADF warstwa monitorowania i zarządzania podzielona została na dwie części: związaną ze środowiskiem wykonawczym i związaną z aplikacją. W pracy skoncentrowano się na przedstawieniu części związanej z monitorowaniem i zarządzaniem aplikacją, ponieważ do monitorowania środowiska wykorzystano model CIM i dostępną infrastrukturę WBEM.

[5] W pracach wykorzystano wirtualizację dostępną w systemie Solaris 10.



Rysunek 9: Diagram sekwencji pokazujący interakcję pomiędzy menadżerem adaptacji `AdaptationManager` i infrastrukturą prostego rozmieszczania; część (a) pokazuje aktualizację wewnętrzną — etap przygotowania środowiska do uruchomienia komponentów; część (b) przedstawia aktualizację zewnętrzną — migrację komponentów

Projekt i implementacja mechanizmów monitorowania i zarządzania aplikacją, a w szczególności mechanizmu migracji komponentów w czasie wykonania, stanowi jedno z głównych osiągnięć niniejszej pracy. Przy realizacji tej infrastruktury przyjęto założenie, że techniką ponownego rozmieszczania będzie rozmieszczanie w czasie wykonania, a mechanizmami realizacji tej techniki będą migracja komponentów i monitorowanie komunikacji z użyciem intercepcji. Praca koncentruje się na ponownym rozmieszczaniu w czasie wykonania, ponieważ:

- jest to technika, która może zagwarantować najbardziej „zwinny" (ang. *agile*[6]) system adaptacji,
- jej duża szybkość odpowiedzi umożliwia zastosowanie wyszukanych algorytmów planowania rozmieszczenia w czasie wykonania,
- stworzenie mechanizmów migracji i intercepcji jest interesującym zadaniem i stanowi wyzwanie.

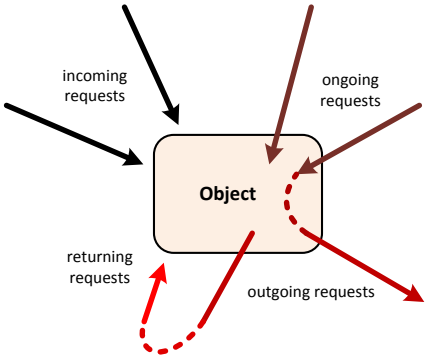
5.1 Mechanizm migracji komponentów w czasie wykonania

Przy projektowaniu i implementacji mechanizmu migracji komponentów w czasie wykonania przyjęto założenia, które miały bardzo istotny wpływ na jego realizację:

- migracji podlegają komponenty CCM, które dopuszczają wykonanie wielowątkowe. Przez to zagadnienie migracji staje się trudne, ale bardziej użyteczne biorąc pod uwagę obecne trendy w architekturze procesorów wielordzeniowych,
- mechanizm ma gwarantować migrację słabą (w odróżnieniu od migracji silnej [10]). Rozwiązanie migracji silnej w przypadku komponentów warstwy pośredniej wydaje się niemożliwe w implementacji,
- mechanizm migracji ma być możliwie przeźroczysty dla programisty, przy jednoczesnym zachowaniu dużej swobody w implementacji komponentów CCM,
- jednym z istotnych kryteriów oceny migracji będzie czas wstrzymania komponentu potrzebny do jego przeniesienia pomiędzy lokalizacjami. Wstrzymanie pracy pojedynczego komponentu może znacząco wpływać na działanie całej aplikacji.

Migracja komponentów CCM jest bezpośrednio związana z migracją obiektów CORBA. Analiza problemu migracji obiektów prowadzi do wyróżnienia następujących jej etapów: (1) zawieszenia, (2) zapisu stanu, (3) transferu stanu, (4) wczytania stanu, (5) przełączenia, (6) aktywacji i (7) usunięcia. W heterogenicznych systemach rozproszonych z wielowątkowymi komponentami rozwiązanie niemal każdego z wymienionych etapów jest zagadnieniem trudnym. Dlatego, aby ograniczyć nieco

[6] Zwinność adaptacji (ang. *adaptation agility*) to termin sformułowany przez B.D. Noble'a i in. w [17], który określa szybkość i dokładność z jaką potrafi ona podążać za zmianami w kontekście wykonania.



Rysunek 10: Cztery możliwe przypadki związane z obsługą żądań, które wymagają uwzględnienia podczas zawieszenia obiektu

zakres problemu migracji, w pracy pominięto problem transmisji stanu w przypadku środowisk heterogenicznych. Przyjęto wspólną platformę implementacji modelu CCM opartą o język Java. Upraszcza to zapis, transfer i wczytanie stanu komponentu i uniezależnia je od platformy sprzętowej i systemu operacyjnego. Szczegółowo przedstawiona została natomiast analiza i rozwiązanie problemów zawieszenia i przełączenia komponentu.

Do poprawnego zawieszenia komponentu/obiektu wymagane jest jego przejście w stan spoczynku (ang. *quiescent state* [14]). Osiągnięcie tego stanu możliwe jest dopiero po właściwej obsłudze czterech typów żądań; ilustruje to rys. 10: żądań przychodzących, żądań w trakcie obsługi, żądań wychodzących i przypadku najtrudniejszego — żądań wracających tj. żądań przychodzących w wyniku synchronicznych wywołań wychodzących. W pracy omówiono sposoby obsługi dla wszystkich czterech przypadków, a implementacja i ocena działania mechanizmu migracji pozwoliła zweryfikować poprawność tych propozycji.

Przełączenie komponentu w środowisku rozproszonym jest niemniej istotnym i trudnym zadaniem. Napotyka ono na problem zależności szczątkowych (ang. *residual dependency problem*), który określa na ile komponent przeniesiony do lokalizacji docelowej zależy od lokalizacji źródłowej. Zaproponowane w pracy rozwiązanie nawiązuje do podejścia znanego z protokołu MobileIP [9] i wykorzystuje technikę z agentem domowym. Stanowi ona kompromis pomiędzy praktycznie nieosiągalną aktualizacją wszystkich łączy prowadzących do migrującego komponentu, a bardzo wrażliwym na awarie podejściem z łańcuchem odwołań. Rozwiązanie z agentem domowym bardzo dobrze wpisuje się w model CCM, w którym komponenty mogą być zarządzane przez infrastrukturę rozmieszczania lub przez fabrykę. W pracy pokazano jak w naturalny sposób funkcje agenta domowego mogą zostać przejęte przez fabrykę komponentów.

Prace nad mechanizmem migracji doprowadziły do wniosku, że opracowanie w pełni przeźroczystego mechanizmu migracji komponentów na poziomie warstwy pośredniej (ang. *middleware*) wydaje się zadaniem bardzo trudnym do osiągnięcia. W

przypadku modelu CCM wymagałoby ono wprowadzenia znacznych ograniczneń na architekturę komponentu, jego implementację i sposób dostępu do zasobów. Dlatego też celem autora było opracowanie mechanizmu, który ułatwiałby tworzenie komponentów mobilnych, a jednocześnie pozwalałby zachować duży potencjał modelu CCM. W tym celu rozszerzono cykl życia komponentu. Z jednej strony angażuje to programistę w zagadnienia mobilności, ale z drugiej pozwala na stosunkowo dużą automatyzację migracji. Proponowane rozwiązanie stoi w sprzeczności z często przyjętym podejściem, w którym komponent definiowany jest jako jednostka kompozycji z dobrze określonym interfejsem i jedynie jawnymi zależnościami.[7] Zdaniem autora dopuszczenie wyłącznie jawnych zależności komponentu względem jego środowiska wykonania jest założeniem zbyt ograniczającym i niepraktycznym.

5.2 Monitorowanie aplikacji z użyciem interceptorów COPI

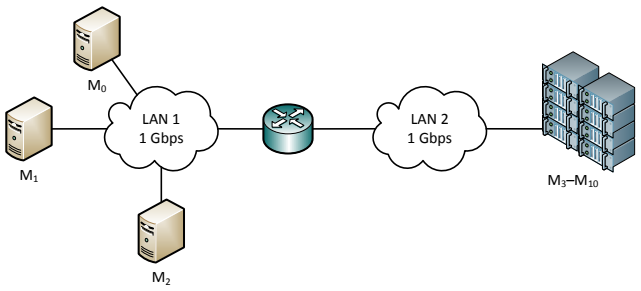
Interceptory stanowią istotny wzorzec projektowy, który znalazł zastosowanie w wielu przypadkach m.in. w szyfrowaniu i filtrowaniu wiadomości, tworzeniu logów komunikatów, a także w monitorowaniu komunikacji. W pracy wykorzystano interceptory do przeźroczystego monitorowania aplikacji. Implementacja mechanizmu została oparta na istniejącej specyfikacji Container Portable Interceptors (COPI) [19]. Jedną z najważniejszych zalet interceptorów COPI, która wynika z komponentowej budowy aplikacji, jest możliwość identyfikacji obu komunikujących się stron tj. komponentu wywołującego i wywoływanego. Wiele z istniejących platform warstwy pośredniej nie daje wprost takiej możliwości (np. .Net WCF czy Java RMI). Umożliwiło to stworzenie m.in. sensora `LinkEffector`, który pozwala określić sumaryczne natężenie komunikacji pomiędzy dwoma komponentami, niezależnie od tego iloma i jakimi portami dane komponenty zostały połączone.

5.3 Problem identyfikacji instancji komponentu

W wyniku prac prowadzonych nad rozmieszczaniem i migracją komponentów oraz intercepcją komunikacji dużego znaczenia nabrała kwestia właściwej identyfikacji instancji komponentu. Przy rozmieszczaniu identyfikacja jest konieczna do właściwego powiązania, aktualizacji i usuwania instancji. Mechanizm intercepcji wymaga identyfikacji komponentów biorących udział w komunikacji. W migracji komponentów właściwa identyfikacja jest konieczna do poprawnego przekierowania żądań.

Z punktu widzenia infrastruktury monitorowania i zarządzania ADF największym wyzwaniem była identyfikacja instancji na potrzeby migracji i intercepcji. Podejście zaproponowane w specyfikacji CCM [20, rozdz. 6] jest w tym wypadku mało efektywne, gdyż do identyfikacji wymaga co najmniej jednego połączenia zdalnego. Drastycznie wpływa to na pogorszenie wydajności komunikacji, dlatego też w ramach prac implementacyjnych zaproponowano własny, niestandardowy mechanizm identyfikacji instancji. Przy jego realizacji użyto identyfikatorów UUID [15], które są osadzane w

[7] Por. C. Szyperski [22].



Rysunek 11: Topologia sieci środowiska testowego

Tablica 1: Konfiguracja sprzętowa i programowa środowiska testowego

Computer ID	M_0	M_1	M_2	$M_3 - M_{10}$
CPU family	Intel Core 2 Duo	Intel Centrino Duo	Pentium 4	UltraSPARC-IIe
CPU clock speed	2.66 GHz	2.6 GHz	2 GHz	650 MHz
Memory size	3 GB	3 GB	1 GB	1 GB
Operating system	Windows Vista	Windows XP	Linux 2.6	Solaris 9

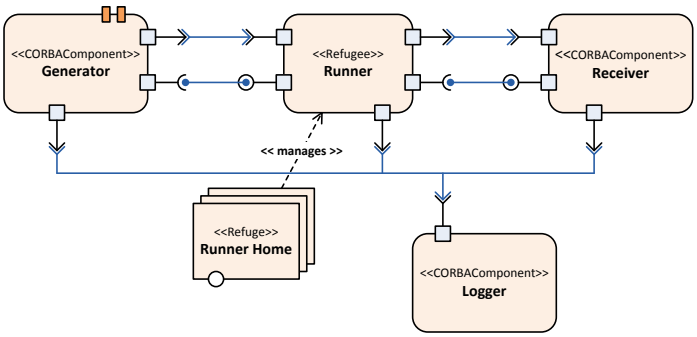
każdej referencji odnoszącej się do komponentu lub jego portu. Tym sposobem do identyfikacji instancji komponentu potrzebna jest jego dowolna referencja i nie ma konieczności wykonywania zdalnych zapytań. Ważne przy tym jest, że identyfikatory instancji raz określone nie ulegają zmianie nawet podczas migracji. Pozwala to zminimalizować narzuty pamięciowe w przypadku wielokrotnej migracji komponentu na tą samą maszynę docelową. W efekcie opracowany mechanizm gwarantuje wysoką wydajność czasową i pamięciową.

6 Ocena składowych platformy ADF

Celem oceny elementów składowych platformy ADF było określenie ich wpływu na działanie aplikacji oraz udziału w całkowitym koszcie adaptacji z użyciem platformy. Wpływ składowych stanowi minimalny koszt adaptacji i jest niezależny od jakości algortymu, który zostanie użyty do adaptacji aplikacji. Ocena została podzielona na trzy części: ocenę infrastruktury prostego rozmieszczania, ocenę mechanizmu migracji oraz ocenę infrasturktury monitorowania.

6.1 Środowisko i aplikacje testowe

Na potrzeby testowania platformy ADF stworzono środowisko laboratoryjne składające się z dwóch sieci lokalnych typu Gigabit Ethernet łączących 11 komputerów (rys. 11). W sieci $LAN1$ pracowały 3 komputery o zróżnicowanych parametrach, natomiast sieć $LAN2$ łączyła klaster 8 serwerów typu $Blade$ o jednolitej konfiguracji sprzętowej i programowej. Parametry maszyn zamieszczono w tabeli 1.



Rysunek 12: Architektura aplikacji testowej: Generator ruchu

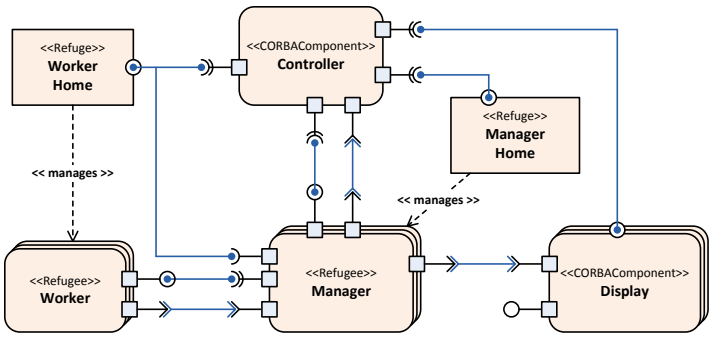
Do oceny działania elementów ADF stworzono także dwie aplikacje testowe:

Generator ruchu — reprezentuje klasę aplikacji z intensywną komunikacją. Aplikacja składa się z pięciu komponentów (rys. 12). Podstawowym komponentem Generatora ruchu jest mobilny komponent `Runner`, który ruch odbierany z `Generatora` przekazuje natychmiast do komponentu odbiorcy `Receiver`. W zależności od konfiguracji, `Generator` może emitować ruch o różnorodnej charakterystyce np. o zadanej częstotliwości czy o rozkładzie losowym.

Asymetryczny generator wizualizacji 3D — jego zadaniem jest symulacja różnych klas systemów. W zależności od konfiguracji Asymetryczny generator wizualizacji 3D (w pracy określany skrótem ART) może symulować aplikacje intensywne obliczeniowo, intensywne komunikacyjnie albo mieszane, w których część aplikacji wykonuje więcej obliczeń, a część przede wszystkim wymienia komunikaty. Architektura aplikacji ART przedstawiona jest na rys. 13. Powstała ona na bazie typowego systemu o architekturze master–worker (tu `Manager–Worker`), który charakteryzuje się dużą symetrią działania. Wprowadzenie dodatkowej warstwy sterowania aplikacją (komponent `Controller`) pozwoliło symulować scenariusze niesymetryczne — aplikacje o mieszanym wzorcu komunikacji.

6.2 Ocena infrastruktury prostego rozmieszczania

Jednym z aspektów ewaluacji infrastruktury prostego rozmieszczania była ocena zgodności implementacji ze specyfikacją D&C. Tabela 2 przedstawia dane ilościowe, a w pracy przedstawiono także szczegóły oceny jakościowej. Na uwagę zasługuje fakt, że model *Execution Management Model*, który jest kluczowym elementem D&C został zaimplementowany w niemal 90% zgodności z tą specyfikacją. Ponadto został on uzupełniony o rozszerzenia umożliwiające adaptację aplikacji w czasie wykonania. Niezaimplementowane fragmenty modelu są wynikiem przyjęcia odmiennego sposobu zarządzania zasobami. W ujęciu proponowanym przez OMG infrastruktura



Rysunek 13: Architektura aplikacji testowej: Asymetryczny generator wizualizacji 3D

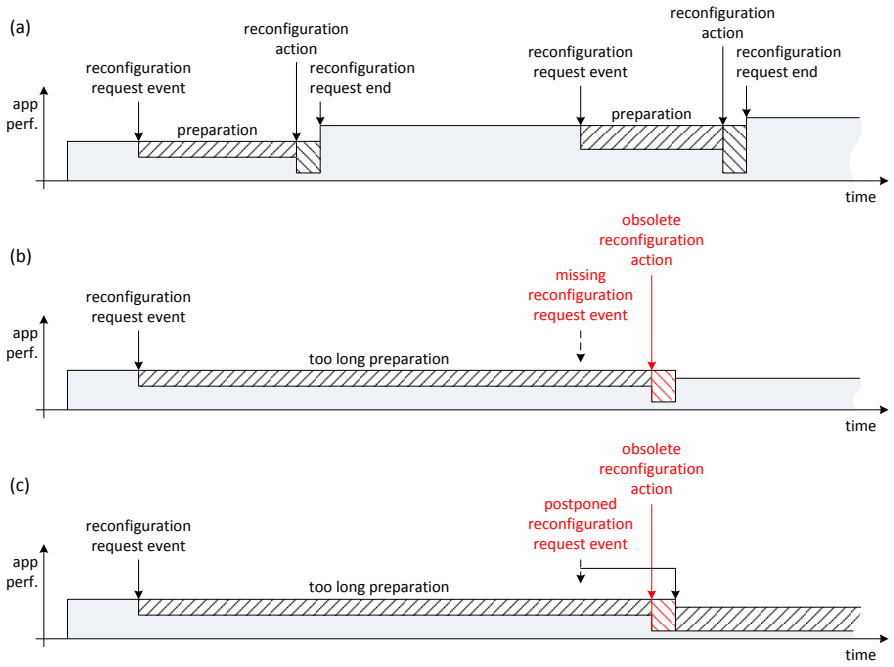
Tablica 2: Liczba i procent interfejsów i ich operacji zdefiniowanych w specyfikacji D&C, które wraz z proponowanymi rozszerzeniami zostały zaimplementowane w infrastrukturze prostego rozmieszczania

Execution Model	No. of entities	No. of operations
Component Management Model	1/1 (100%)	5/8 (63%)
+extensions	0	+1
Target Management Model	1/2 (50%)	2/7 (29%)
Execution Management Model	8/9 (89%)	13/14 (93%)
+extensions	+3	+5
Total	**10/12 (83%)**	**20/29 (69%)**
+extensions	**+3**	**+6**

rozmieszczania powinna dokonywać rezerwacji zasobów, a przez to powinna posiadać wyłączny dostęp do zasobów. Jednym z założeń platformy ADF jest adaptacja w systemach otwartych, do czego w ocenie autora dużo lepiej pasuje zarządzanie zasobami typu *best-effort*.

Przy ocenie infrastruktury prostego rozmieszczania przeprowadzono także badania efektywności jej działania. Miały one na celu weryfikację dwóch aspektów: (1) wpływu etapu przygotowania (rys. 9a) na działającą aplikację oraz (2) czasu potrzebnego na przeprowadzenie etapu przygotowania. Testy wykazały około 1–2% narzut etapu przygotowania na działanie aplikacji, natomiast bezwzględny czas potrzebny na jego przeprowadzenie w dużej mierze zależał o szybkości komputerów i wahał się w granicach 3–22 sekund. Dokładne wyniki tej oceny znajdują się w pracy, poniżej natomiast przedstawiono wnioski.

Etap przygotowania do rekonfiguracji (obejmujący aktywności instalacji i aktywacji) ma istotny wpływ na „zwinność" procesu adaptacji. Im szybsze przygotowanie i rekonfiguracja tym częściej mogą być podejmowane akcje adaptacji. Ponadto fazy przygotowania i rekonfiguracji powinny być zharmonizowane z działaniem algorytmu adaptacji. Problem ten ilustruje rysunek 14, na którym pokazano trzy sytuacje. W



Rysunek 14: Wpływ etapów przygotowania i rekonfiguracji na zwinność procesu adaptacji aplikacji

pierwszej z nich (rys. 14a) procesy przygotowania i rekonfiguracji przebiegają sprawnie i nie zakłócają działania algorytmu adaptacji. Po nadejściu żądania rekonfiguracji (*reconfiguration request event*) infrastruktura prostego rozmieszczania używana jest do przygotowania środowiska wykonawczego. To zużywa zasoby tego środowiska i ma niekorzystny wpływ na wydajność aplikacji. W następnym kroku dokonywana jest rekonfiguracja. W przypadku ADF mechanizmem rekonfiguracji jest migracja komponentów w czasie działania stąd rekonfiguracja przebiega dużo szybciej niż przygotowanie. Niekorzystna sytuacja następuje, gdy żądania rekonfiguracji nadchodzą zbyt szybko (lub etapy przygotowania i rekonfiguracji trwają zbyt długo). Wtedy też mogą wystąpić dwa niekorzystne zjawiska: pominięcie żądania (rys. 14b) lub sztorm rekonfiguracyjny (rys. 14c). Oba obniżają zwykle wydajność aplikacji, gdyż w obu przypadkach podejmowana akcja rekonfiguracji jest nieaktualna. Algorytm adaptacji powinien zatem brać pod uwagę czas i wpływ faz przygotowania i rekonfiguracji jako jeden z parametrów wejściowych.

6.3 Ocena wydajności mechanizmu migracji

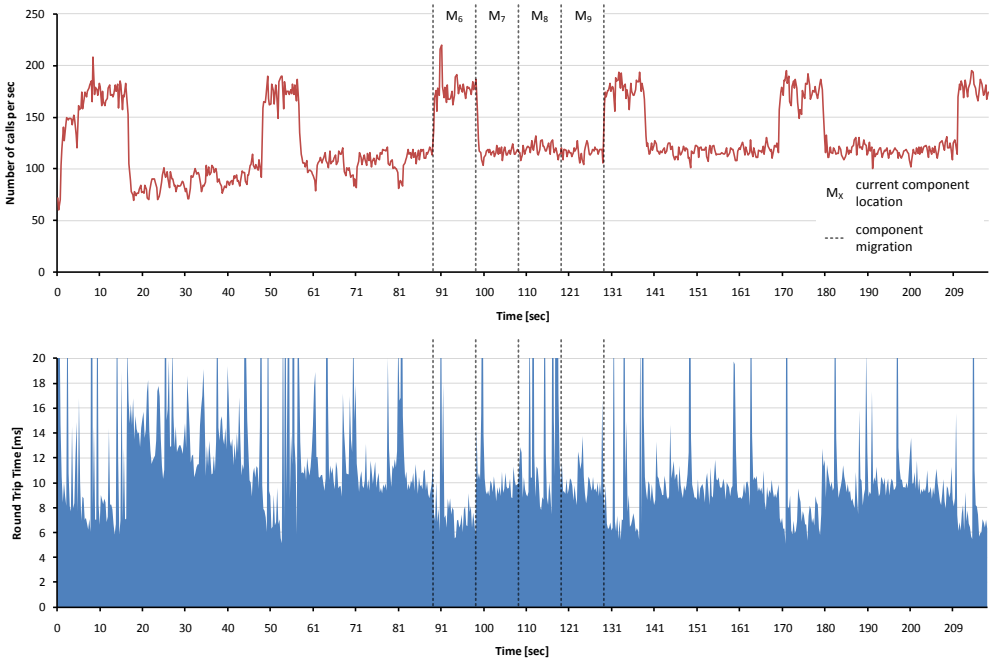
Dokonując oceny wydajności mechanizmu migracji skupiono się na trzech aspektach: (1) bezwzględnym czasie potrzebnym na wykonanie migracji komponentu, (2) wpływie migracji na wydajność aplikacji oraz (3) porównaniu implementacji środowiska CCM uzupełnionego o mechanizm migracji do jego oryginalnej implementacji.

Badania, w których zmierzono czas potrzebny na migrację komponentu zostały wykonane na klastrze serwerów Blade a uzyskany wynik oscylował w granicach 70–80 ms dla komponentu `Runner`. Ponieważ otrzymany wynik zależy od wielu czynników takich jak: szybkość komputerów, liczba portów komponentu, rozmiar stanu, służyć on może jedynie do lepszego zrozumienia relacji fazy przygotowania i migracji. Wynika z tego, że dla platformy ADF migracja jest operacją o co najmniej dwa rzędy wielkości szybszą niż przygotowanie.

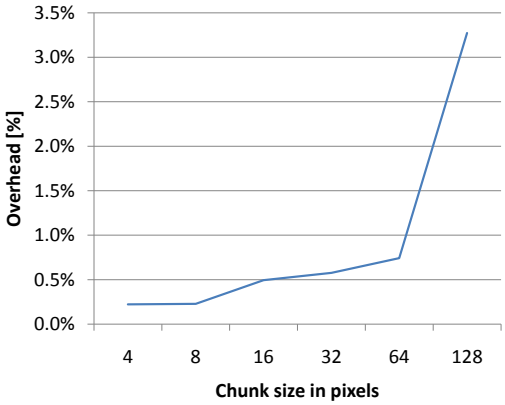
Efektem ubocznym migracji w czasie wykonania jest wstrzymanie pracy komponentu na czas migracji. Określenie dokładnego wpływu takiego wstrzymania na działanie aplikacji jest trudne, gdyż zależy od tego jaką rolę w aplikacji odgrywa dany komponent. Do oceny wpływu migracji na wydajność aplikacji wykonano zatem dwie grupy badań. Pierwsza miała na celu określić wpływ migracji na zdolności komunikacyjne aplikacji Generator ruchu. W tym wypadku mobilny komponent `Runner` jest zasadniczym elementem aplikacji stąd należało oczekiwać stosunkowo dużych zakłóceń w pracy aplikacji. Rysunek 15 przedstawia uzyskane rezultaty; zaznaczono na nim również momenty, w których wykonywana była migracja komponentu. Na rysunku widać duże różnice w liczbie wywołań na sekundę w przypadku, gdy komponent jest w lokalizacji „domowej" w stosunku do pozostałych lokalizacji (wykres górny). W pracy przedstawiono wyjaśnienie tego problemu, który ma związek z implementacją platformy ORB, a nie z jakością implementacji mechanizmu migracji. Ponadto dość dobrze widoczne są momenty, w których czas RTT (ang. *round trip time*) znacząco wzrastał. Do testów nie używano platformy JavaRT stąd czas RTT wzrastał także w innych przypadkach, niemniej jednak z przeprowadzonych badań wynika, że migracja miała niewielki wpływ na aplikację o intensywnej komunikacji.

Druga grupa przeprowadzonych badań miała na celu określenie wpływu migracji na wydajność przetwarzania aplikacji. Do tego celu użyto aplikacji ART z różnym rozmiarem generowanego bloku, co pozwoliło symulować aplikacje bardziej intensywne komunikacyjnie (mniejszy blok) lub bardziej wymagające obliczeniowo (większy blok). Wykres na rysunku 16 przedstawia rezultaty. Wynika z nich, że wpływ migracji komponentu `Worker` jest niewielki (poniżej 1.0%) do momentu, w którym rozmiar bloku jest na tyle duży, że na jego wygenerowanie potrzeba było więcej czasu niż wynosił interwał pomiędzy kolejnymi migracjami. Jak widać, dla niewłaściwie przygotowanej implementacji komponentu zbyt częsta migracja może spowodować zablokowanie aplikacji. Jest to argument za tym, aby podczas tworzenia komponentu programista zdawał sobie sprawę czy implementuje komponent mobilny czy niemobilny. Problem wpływu migracji na wydajność przetwarzania został dokładniej omówiony w pracy; porównano tam cztery przypadki: migrację silną, migrację słabą z utratą stanu, migrację słabą z wstrzymaniem oraz migrację słabą z selektywnym odrzuceniem żądań.

Ostatnią grupą badań oceniających jakość mechanizmu migracji było porównanie wydajności aplikacji działającej przy wykorzystaniu oryginalnej implementacji platformy CCM oraz platformy uzupełnionej o elementy migracji. Podczas badań nie wykonywano żadnych migracji, tak aby można było porównać czy wersja mobilna komponentów `Worker` i `Manager` wpływa niekorzystnie na wydajność. Uzyskane



Rysunek 15: Wpływ migracji komponentu `Runner` na liczbę wywołań w jednostce czasu oraz czas RTT wywołania



Rysunek 16: Narzuty na wydajność aplikacji ART związane z migracją komponentu `Worker`

Tablica 3: Czas wykonania aplikacji ART (w sekundach) działającej z oryginalną i zmodyfikowaną wersją platformy CCM

Application settings	Original OpenCCM	Mobility-aware OpenCCM	Overhead
comm.-intensive	$472.1(\pm 0.3\%)$	$463.0(\pm 0.3\%)$	-1.9%
CPU-intensive	$304.3(\pm 0.1\%)$	$291.9(\pm 1.0\%)$	-4.1%
mixed	$515.8(\pm 0.8\%)$	$516.7(\pm 1.8\%)$	0.2%

Tablica 4: Czas wykonania aplikacji ART (w sekundach) w zależności od uruchomienia infrastruktury WBEM i sensorów CIM

Application settings	WBEM disabled	WBEM enabled				
		no sensors	`CPU-Sensor`	`Memory-Sensor`	`CPU-` and `MemorySensor`	`Null-Sensor`
CPU-intensive	$290.0(\pm 0.8\%)$	$290.6(\pm 0.6\%)$	$349.6(\pm 0.8\%)$	$349.9(\pm 1.5\%)$	$415.0(\pm 0.5\%)$	$293.3(\pm 1.6\%)$
comm.-int.	$429.7(\pm 0.6\%)$	$430.1(\pm 1.0\%)$	$521.3(\pm 0.7\%)$	$521.3(\pm 1.3\%)$	$611.0(\pm 0.3\%)$	$438.2(\pm 0.4\%)$
mixed	$514.7(\pm 0.8\%)$	$516.8(\pm 1.2\%)$	$616.7(\pm 0.5\%)$	$617.1(\pm 1.5\%)$	$735.9(\pm 0.6\%)$	$526.8(\pm 0.8\%)$

wyniki przedstawiono w tabeli 3. Wynika z nich, że wprowadzenie mobilności komponentów nie musi wiązać się ze spadkiem wydajności środowiska. Stanowi to dodatkową motywację do prowadzenia dalszych badań w tym zakresie.

6.4 Narzuty infrastruktury monitorowania

Infrastruktura monitorowania jest zasadniczą częścią każdego środowiska adaptacyjnego stąd bardzo istotne jest określenie (negatywnego) wpływu jaki ma ona na działanie aplikacji. W celu minimalizacji tego wpływu platforma ADF może dokonywać instalacji i deinstalacji sensorów na żądanie tak, aby działały one jedynie w tych obszarach systemu, które są w danym momencie najbardziej istotne dla algorytmu adaptacji. Należy przy tym zauważyć, że sensory stanowią jedynie fasadę dla odpowiednich mechanizmów niższego poziomu (warstwy instrumentacji), które dokonują właściwej obserwacji systemu. O ile instalacja i deinstalacja sensorów jest możliwa, o tyle ADF nie pozwala na włączanie i wyłączanie elementów warstwy instrumentacji.

W przeprowadzonych badaniach pokazano wpływ obu czynników: włączenia i wyłączenia elementów warstwy instrumentacji oraz, włączenia i wyłączenia sensorów. Badania podzielono na trzy części: (1) ocenę sensorów środowiska wykonawczego wykorzystujących model CIM i infrastrukturę WBEM, (2) sensorów aplikacji wykorzystujących interceptory COPI oraz (3) sensora warstwy aplikacji korzystającego z mechanizmów dostępnych w platformie OpenCCM. Wyniki badań przedstawiono w tabelach 4, 5, 6. Jak widać bardzo niekorzystny wpływ na działanie systemu miała aktywacja infrastruktury WBEM. Po jej uruchomieniu czas wykonania aplikacji wzrastał o 20% dla jednego sensora i aż o 40% dla dwóch sensorów. Wpływ pozostałych sensorów na działanie aplikacji był w granicach 1–8% co jest wartością akceptowalną.

Tablica 5: Czas wykonania aplikacji ART (w sekundach) w zależności od uruchomienia infrastruktury COPI i sensorów wykorzystujących interceptory

Application settings	COPI disabled	COPI enabled			
		no sensors	`Link-Sensor`	`Comm-Sensor`	`Link-` and `CommSensor`
CPU intensive	$290.0(\pm 0.8\%)$	$290.9(\pm 0.2\%)$	$293.6(\pm 0.6\%)$	$299.6(\pm 0.7\%)$	$308.5(\pm 1.0\%)$
comm. intensive	$429.7(\pm 0.6\%)$	$463.6(\pm 0.6\%)$	$476.3(\pm 0.7\%)$	$471.4(\pm 0.4\%)$	$480.9(\pm 0.4\%)$
mixed	$514.7(\pm 0.8\%)$	$529.7(\pm 1.0\%)$	$527.0(\pm 0.9\%)$	$543.4(\pm 1.9\%)$	$543.0(\pm 1.1\%)$

Tablica 6: Czas migracji komponentu `Runner` (w milisekundach) w zależności od uruchomienia sensora instancji `HomeSensor`

Application settings	`HomeSensor` disabled	`HomeSensor` enabled	Overhead
1000 jumps, best-effort	$63.7(\pm 20.0\%)$	$68.7(\pm 19.7\%)$	7.8%

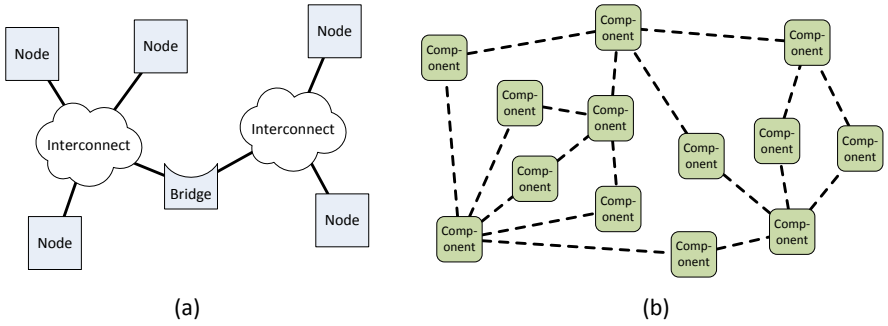
7 Rozmieszczanie adaptacyjne z użyciem algorytmu FDA

Zasadniczym celem platformy rozmieszczania adaptacyjnego ADF jest adaptacja aplikacji poprzez zmianę rozmieszczenia komponentów w środowisku wykonawczym. Koncepcja algorytmu adaptacji zrodziła się z obserwacji, że zarówno komponenty aplikacji jak i elementy środowiska wykonawczego mogą być reprezentowane w formie grafu, a rozmieszczenie komponentów w środowisku to skojarzenie węzłów grafu aplikacji z węzłami grafu tego środowiska. W efekcie zauważono, że do rozmieszczania oprogramowania można zastosować algorytmy rozplanowania grafów (ang. *graph layout algorithms*), a w szczególności algorytmy typu Force-Directed Algorithm (FDA) [24].

Jednym z kluczowych podejść w realizacji algorytmów FDA jest symulacja układu N ciał. W podejściu tym pomiędzy wszystkimi ciałami układu działają siły odpychania (np. siła Coulomba pomiędzy cząstkami naładowanymi jednoimiennie), a pomiędzy wybranymi elementami układu — siły przyciągania (np. symulowane przez sprężyny). W takim ujęciu zadaniem algorytmu FDA jest minimalizacja energii kinetycznej układu. Zakłada się przy tym, że jej minimalizacja powala uzyskać pożądany rozkład wierzchołków w grafie.

W pracy zaproponowano algorytm planowania rozmieszczenia FDDP jako rozwiązanie problemu planowania w czasie działania (w odróżnieniu od statycznego planowania początkowego). Wykorzystano przy tym iteracyjną naturę podejścia FDA, które dobrze pasuje do systemu dynamicznego jakim jest aplikacja uruchomiona w otwartym środowisku rozproszonym. Mimo, że FDDP nie stanowi zasadniczego wkładu pracy jest nowatorską propozycją na rozwiązanie problemu rozmieszczania w środowiskach rozproszonych.

W FDDP aplikacja jest modelowana jako graf, w którym wierzchołki reprezentują



Rysunek 17: Przykłady grafów reprezentujących w FDDP: (a) środowisko wykonawcze i (b) aplikację komponentową

komponenty aplikacji, a krawędzie — aktywne połączenia pomiędzy komponentami (rys. 17b). Środowisko wykonawcze to graf komputerów połączonych siecią. Do modelowania środowiska wykorzystano bezpośrednio model *Target Data Model* specyfikacji D&C, na który składają się trzy zasadnicze elementy (rys. 17a): węzły wykonawcze (`Node`), elementy sieciowe (`Interconnect`) i elementy łączące (`Bridge`). Wszystkie trzy elementy są w FDDP reprezentowane jako wierzchołki grafu natomiast krawędzie to połączenia sieciowe pomiędzy nimi. Taka reprezentacja pozwala w prosty sposób zamodelować sieci komputerowe: `Node` może reprezentować komputer, `Interconnect` — przełącznicę, a `Bridge` — ruter sieciowy. Rozmieszczenie oprogramowania jest w FDDP modelowane za pomocą krawędzi łączących wierzchołki obu grafów. Krawędzie występują pomiędzy wierzchołkami komponentów aplikacji i węzłów wykonawczych, na których komponenty aktualnie się wykonują. W takim ujęciu zadaniem FDDP jest rozplanowanie podgrafów aplikacji i środowiska wykonawczego, a następnie znalezienie odwzorowania pomiędzy nimi.

7.1 Dobór sił w algorytmie FDDP

Do poprawnego działania algorytmu FDDP kluczowe jest właściwe odwzorowanie relacji panujących w rzeczywistym systemie rozproszonym na siły w układzie N ciał, na którym operuje algorytm. W pracy przedstawiono szczegóły zaproponowanego odwzorowania. Obejmuje ono sześć sił:

- R_{nn} — siła odpychania pomiędzy węzłami, której celem jest rozplanowanie węzłów środowiska wykonawczego; w FDDP modelowana przez siłę Coulomba działająca na ładunki jednoimienne,
- A_{ni} — siła przyciągania pomiędzy węzłami wykonawczymi i elementami sieciowymi, której celem jest zgrupowanie węzłów połączonych dobrej jakości siecią; w FDDP modelowana przez siłę Hooke'a oddziałującą pomiędzy dwoma ciałami połączonymi sprężyną,

- R_{cc} — siła odpychania pomiędzy komponentami odpowiedzialna za rozplanowanie komponentów aplikacyjnych; w FDDP modelowana przez siłę Coulomba działającą na ładunki jednoimienne,
- A_{cc} — siła przyciągania pomiędzy komunikującymi się komponentami; jej celem jest zgrupowanie komponentów intensywnie się komunikujących; w FDDP modelowana przez siłę Hooke'a,
- A_{ch} — siła przyciągania pomiędzy komponentem i jego aktualnym węzłem wykonawczym, która symuluje koszt migracji komponentu pomiędzy węzłami; w FDDP modelowana przez siłę Hooke'a,
- A_{cn} — siła przyciągania pomiędzy komponentami i wszystkimi węzłami, która jest proporcjonalna do wielkości zasobów dostępnych na węźle wykonawczym; jej zadaniem jest właściwe rozplanowanie rozmieszczenia komponentów aplikacji; w FDDP modelowana przez siłę grawitacji.

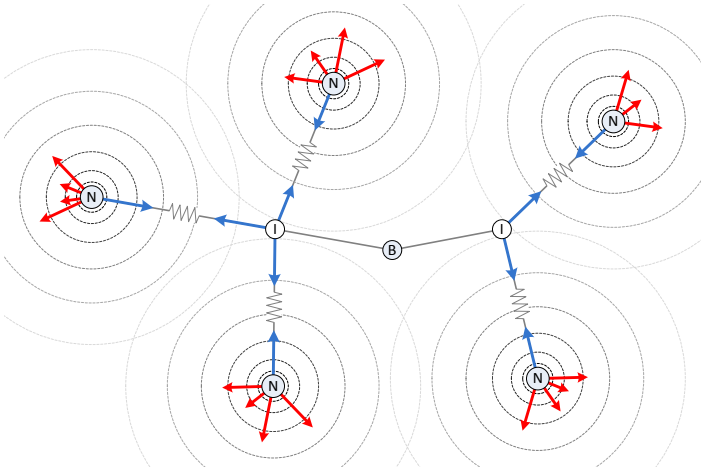
Rysunek 18 ilustruje działanie dwóch z wymienionych sił: R_{nn} i A_{ni} w układzie pięciu oddziałujących na siebie węzłów połączonych dwoma elementami `Interconnect` i jednym `Bridge`.

Omówione powyżej odwzorowanie sił jest jednym z bardzo wielu możliwych. Do poprawnego działania, FDDP wymaga także określenia wielu stałych, których wartości niekiedy istotnie wpływają na jego zachowanie. Ustalenia parametrów modelu umożliwiających stabilną pracę algorytmu dokonano w trakcie prac eksperymentalnych. Pozwoliło to również określić zależności pomiędzy wybranymi parametrami modelu a zachowaniem FDDP istotnym z punktu widzenia użytkownika. Przykładowo, aby komponenty aplikacyjne zostały równomiernie rozmieszczone w środowisku wykonawczym, należy zmniejszyć wpływ siły A_{cc} i zwiększyć działanie siły A_{cn}. W pracy przedstawiono także inne zależności, dzięki którym algorytm FDDP może być kontrolowany przez użytkownika w celu realizacji wybranych strategii adaptacji.

7.2 Ocena ADF

Przy ocenie działania i skuteczności platformy ADF uwagę zwrócono na trzy aspekty: (1) określenie całkowitego kosztu adaptacji przez platformę ADF włączając w to wszystkie jej elementy, (2) określenie wpływu adaptacji na wydajności aplikacji oraz (3) ocenę zachowania adaptacji w przypadku pojawienia się zakłócenia zewnętrznego w środowisku wykonawczym.

Do oceny kosztów działania platformy ADF wykorzystano cztery identyczne maszyny typu *Blade* oraz aplikację ART. Test został tak skonstruowany, aby ADF dokonało dynamicznego rozmieszczenia infrastruktury wykonawczej komponentu i sensorów monitorujących oraz wykonało migrację jednego, wybranego komponentu. Ponieważ komputery miały identyczną konfigurację sprzętową i programową, działanie platformy ADF wprowadzało jedynie narzuty na wydajność przetwarzania. Rezultaty



Rysunek 18: Ilustracja sił odpychania R_{nn} (strzałki czerwone) i przyciągania A_{ni} (strzałki niebieskie) pomiędzy wierzchołkami grafu reprezentującego środowisko wykonawcze w FDDP

Tablica 7: Czasy wykonania aplikacji ART (w sekundach) przy włączonej i wyłączonej infrastrukturze ADF

		With ADF enabled		
Chunk size	No ADF at home	At home no moves	At non-home 1 move	10 moves
10×10	117.0(±1.3%)	151.9(±1.2%)	184.2(±0.1%)	196.5(±1.9%)
32×32	74.0(±2.0%)	92.9(±1.5%)	107.9(±0.9%)	117.3(±0.6%)

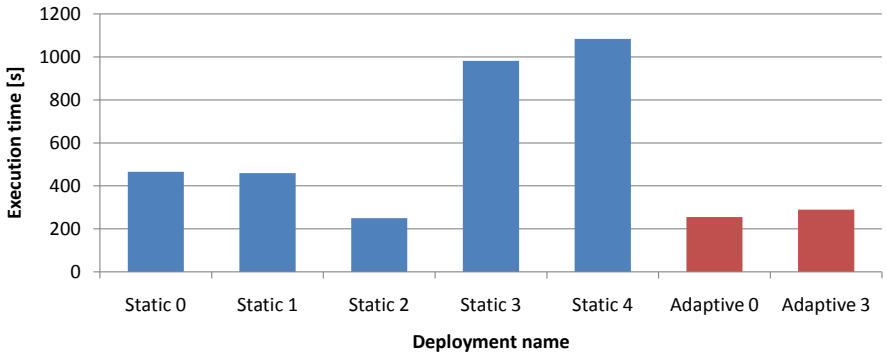
testu przedstawiono w tabeli 7. Wynika z nich, że samo uruchomienie platformy wydłuża czas wykonania aplikacji o około 30%. W przypadku, migracji wybranego komponentu narzuty na czas wykonania wzrosły do blisko 70%. Warto jednak zwrócić uwagę, że celem przygotowanego testu było pokazanie jednego z najbardziej niekorzystnych scenariuszy, w którym migracja komponentu wiązała się z wstrzymaniem działania całej aplikacji. Ponadto implementacja tego komponentu realizowała podejście ze stratą stanu,[8] które dodatkowo zwiększa narzuty związane z migracją.

Druga grupa eksperymentów miała za zadanie określenie wpływu adaptacji na wydajność aplikacji ART. W tym celu porównano działanie aplikacji przy statycznie zaplanowanym rozmieszczeniu jej komponentów z aplikacją zarządzaną przez ADF. Wyniki prezentuje tabela 8 i rysunek 19. Ze zbioru wszystkich możliwych rozmieszczeń statycznych wybrano tylko takie, które wydawały się dawać najlepsze rezultaty. Mimo to łatwo zauważyć, że wyniki uzyskane dla aplikacji zarządzanej przez ADF (*Adaptive 0* i *Adaptive 3*) były porównywalne z najlepszym znalezionym rozmieszczeniem statycznym (*Static 2*). Pokazuje to, że mimo stosunkowo wysokich kosztów

[8] Por. praca s. 145–146 i rys. 6.12b.

Tablica 8: Czasy wykonania aplikacji ART dla wybranych statycznych i początkowych rozmieszczeń komponentów

Deployment name	Manager A 4 Workers	Manager B 5 Workers	Manager C 4 Workers	Execution time [s]
Static 0	M_0	M_2	M_3	$465.7(\pm 1.0\%)$
Static 1	M_0	M_2	M_4	$459.3(\pm 1.1\%)$
Static 2	M_2	M_0	M_4	$250.1(\pm 0.5\%)$
Static 3	M_3	M_0	M_2	$981.6(\pm 1.2\%)$
Static 4	M_0	M_3	M_2	$1083.0(\pm 1.0\%)$
Adaptive 0	M_0	M_2	M_3	$254.8(\pm 12.5\%)$
Adaptive 3	M_3	M_0	M_2	$289.9(\pm 16.2\%)$

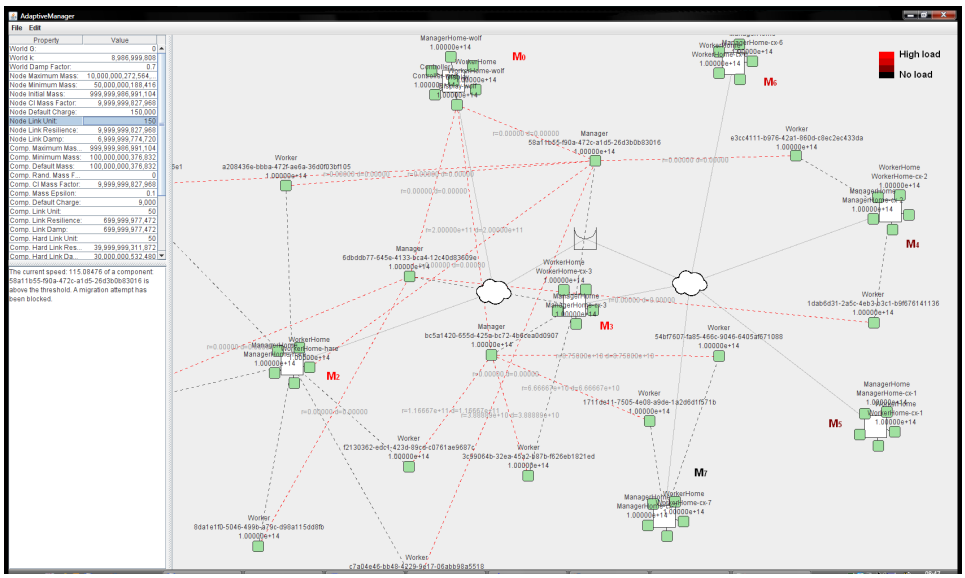


Rysunek 19: Czasy wykonania aplikacji ART dla wybranych statycznych i początkowych rozmieszczeń komponentów

działania prototypowej implementacji platformy ADF, adaptacja aplikacji poprzez zmianę rozmieszczenia komponentów może bardzo korzystnie wpływać na wydajność jej działania. W pracy przedstawiono uzasadnienie dla uzyskanych wyników.

Dane zebrane w tabeli pokazują także, jaki wpływ na wydajność ma początkowe rozmieszczenie komponentów. Różnice w czasie wykonania aplikacji dla rozmieszczeń początkowych *Adaptive 0* i *Adaptive 3* wiążą się z kosztem ich doprowadzenia do stanu bardziej korzystnego. Im gorsze rozmieszczenie początkowe tym większy koszt takiej rekonfiguracji.

Trzecią grupą badań objęto wykonanie aplikacji w przypadku pojawienia się zewnętrznego zakłócenia, którego można się spodziewać w otwartym systemie rozproszonym. Do tego celu ponownie wykorzystano aplikację ART, ale tym razem uruchomione zostały jej dwie instancje. Jedna — obserwowana — której wydajność mierzono, została uruchomiona przy statycznym zaplanowaniu rozmieszczenia lub była zarządzana przez ADF. Druga rozmieszczona statycznie, pełniła rolę zakłócenia. Podczas testów platforma ADF wykazywała pożądane zachowanie. Rysunki 20 i 21 przedstawiają stan przed pojawieniem się zakłócenia w środowisku i stan na kilkanaście sekund po pojawieniu się zakłócenia; niemal wszystkie komponenty mobilne



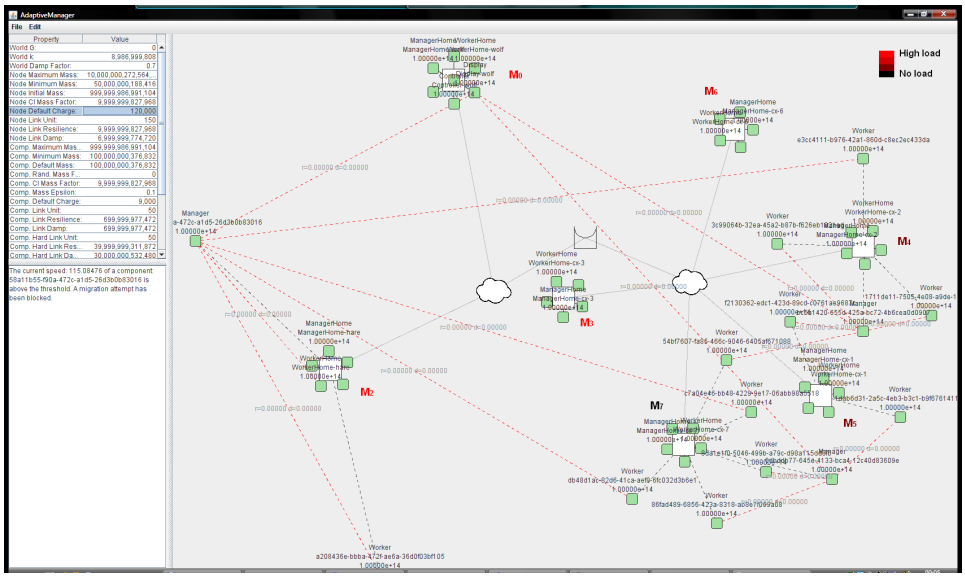
Rysunek 20: Rozmieszczenie komponentów w pierwszej fazie wykonania — przed pojawieniem się zewnętrznego zakłócenia

zostały przeniesione na węzły o największej dostępnej mocy obliczeniowej. W tabeli 9 zebrano czasy wykonania obserwowanej instancji ART dla rozmieszczeń statycznych i zarządzanego przez ADF. Podobnie jak poprzednio przedstawione wyniki pokazują bardzo korzystny wpływ adaptacji na działanie aplikacji. Tym razem czas wykonania był o 20% krótszy niż czas dla najlepszego znalezionego statycznego rozmieszczenia komponentów. W pracy przedstawiono wyjaśnienia, dla których możliwe było osiągnięcie tak korzystnego rezultatu.

7.3 Ograniczenia FDDP

Ocena działania platformy ADF pozwala stwierdzić, że algorytm FDDP spełnił oczekiwania i pozwolił na efektywne planowanie rozmieszczenia w czasie działania. W wyniku prac nad algorytmem FDDP napotkano jednak na pewne trudności w jego zastosowaniu. Są one efektem następujących czynników:

- trudności w odwzorowaniu parametrów modelu na zmienne obserwowane w systemie,
- niejasnych zależności pomiędzy parametrami modelu środowiska wykonawczego i parametrami modelu aplikacji,
- problemów ze skalowalnością i stabilnością rozwiązania,
- niejasną kontrolą algorytmu przez użytkownika.



Rysunek 21: Rozmieszczenie komponentów po pojawieniu się na węzłach M_0, M_2, M_3 i M_6 głównych źródeł zakłócenia

Tablica 9: Czasy wykonania aplikacji ART (w sekundach) dla wybranych rozmieszczeń statycznych w porównaniu z rozmieszczeniem zarządzanym przez ADF

Deployment	`Managers` locations	`Workers` locations	Execution time
Static 0	{A, B, C} $\Rightarrow M_2$	{A, B, C} $\Rightarrow M_0$	$519.8(\pm 0.7\%)$
Static 1	A $\Rightarrow M_5$, B $\Rightarrow M_4$ C $\Rightarrow M_3$	{A, B} $\Rightarrow M_2$ C $\Rightarrow M_6$	$448.8(\pm 0.9\%)$
Static 2	{A, C} $\Rightarrow M_5$ B $\Rightarrow M_4$	{A, B, C} $\Rightarrow M_2$	$590.7(\pm 0.6\%)$
Static 3	A $\Rightarrow M_2$, B $\Rightarrow M_4$ C $\Rightarrow M_5$	{A, B} $\Rightarrow M_2$ C $\Rightarrow M_6$	$454.0(\pm 0.6\%)$
Adaptive 4	{A, B, C} $\Rightarrow M_2$	{A, B, C} $\Rightarrow M_2$	$353.1(\pm 9.1\%)$

Powodują one, że podejście proponowane przez algorytm FDDP z pewnością wymaga dalszych badań. Biorąc jednak pod uwagę wyniki uzyskane przy ocenie efektywności działania platformy ADF można stwierdzić, że jest to podejście obiecujące. Mimo ograniczeń i stosunkowo dużych narzutów prototypowej implementacji ADF wyniki pokazują wysoką skuteczność planowania z użyciem FDDP. Jednocześnie użycie tego algorytmu pozwala zauważyć potencjał skonstruowanej platformy rozmieszczania adaptacyjnego.

8 Wnioski końcowe

W pracy przedstawiono platformę ADF, która pozwala na adaptację aplikacji uruchomionej w rozproszonym środowisku o heterogenicznych zasobach. Adaptacja ta dokonuje się na drodze zmiany rozmieszczenia komponentów aplikacji. Stworzenie prototypu platformy jest podstawowym osiągnięciem autora i jest bezpośrednio związane z wykazaniem tezy rozprawy. Platforma dedykowana jest dla aplikacji komponentowych zbudowanych w oparciu o środowisko warstwy pośredniej CCM. To klasyfikuje ją pomiędzy rozwiązaniami opartymi o wirtualizację systemową a rozwiązaniami na poziomie obiektów. Prowadzenie adaptacji poprzez zmianę rozmieszczenia komponentów wydaje się słuszne co najmniej z dwóch powodów. Po pierwsze oprogramowanie komponentowe z definicji powinno podlegać rozmieszczaniu, co w połączeniu z automatyzacją planowania otwiera drogę adaptacji. Po drugie zmiana rozmieszczenia komponentów może być efektywna, gdyż są one zwykle znacznie mniejsze niż obrazy systemu operacyjnego pozwalając na szybszą rekonfigurację. Jednocześnie nie są tak drobnoziarniste jak obiekty, a to zmniejsza narzuty związane z planowaniem i ponownym rozmieszczaniem.

Podstawą działania zaprezentowanej platformy ADF jest infrastruktura prostego rozmieszczania. W pracy zaproponowano zaawansowany model rozmieszczania, którego wybrane elementy zostały zrealizowane. Budowa pełnej implementacji tego modelu dla rozproszonych systemów komponentowych o heterogenicznych zasobach to zadanie bardzo złożone. Stąd też implementacja ADF obejmuje tylko te aspekty modelu, które pozwoliły na stworzenie i weryfikację procesu rozmieszczania adaptacyjnego. Jednym z takich aspektów jest rozwiązanie trudnego problemu planowania. W pracy pokazano w jaki sposób dzięki realizacji ponownego rozmieszczania można podzielić go na dwie fazy: planowanie początkowe i planowanie w czasie działania. W opisanym rozwiązaniu plan początkowy jest poprawiany po uruchomieniu aplikacji, w miarę napływu bieżących informacji o jej działaniu i działaniu środowiska wykonawczego.

Jednym z kluczowych osiągnięć pracy jest projekt i implementacja mechanizmu migracji komponentów w czasie wykonania. Okazał się on być wystarczająco szybkim i „lekkim” narzędziem rekonfiguracji przez co mógł stać się podstawą realizacji techniki ponownego rozmieszczania w czasie wykonania a w efekcie adaptacji aplikacji. Należy jednak zauważyć, że opracowanie migracji w środowisku komponentów CCM, które pozwala na ich asynchroniczne i wielowątkowe działanie, jest zadaniem

niełatwym. Wymusza to rozwiązanie wielu podstawowych problemów takich jak: przechowywanie stanu, problem zależności szczątkowych, czy osiągnięcie stanu spoczynku. Ponieważ w takim środowisku trudno o całkowitą przeźroczystość bez nałożenia znaczących ograniczeń na budowę komponentu, w pracy zaproponowano podejście, które angażuje programistę w przygotowanie implementacji do migracji. Uzyskano to poprzez rozszerzenie cyklu życia komponentu. Zdaniem autora takie ujęcie z jednej strony ułatwia tworzenie komponentów mobilnych, a z drugiej pozwala w pełni wykorzystać możliwości oferowane przez mechanizm migracji i model CCM.

W heterogenicznym środowisku rozproszonym do poprawnej migracji komponentów wymagane jest wsparcie ze strony infrastruktury rozmieszczania. Jej zadaniem jest w tym przypadku weryfikacja poprawności żądań rekonfiguracji i ewentualne przygotowanie środowiska wykonawczego na zmiany. Z drugiej strony dzięki migracji możliwa jest realizacja ponownego rozmieszczania, a przez to uproszczenie rozwiązania problemu planowania. Wynika z tego, że migracja komponentów w czasie działania i ich rozmieszczanie są mechanizmami komplementarnymi. Połączone razem ułatwiają nie tylko proces uruchomienia aplikacji, ale także pozwalają zwiększyć jej wydajność. W pracy pokazano, że pomimo stosunkowo wysokich kosztów użycia prototypowej platformy ADF, zastosowanie rozmieszczania adaptacyjnego bardzo korzystnie wypływa na wydajność aplikacji. Z przedstawionych badań eksperymentalnych wynika, że dla pewnych klas aplikacji dzięki rozmieszczaniu adaptacyjnemu można osiągnąć rezultaty porównywalne z wynikami uzyskanymi przy najlepszym znalezionym statycznym rozmieszczeniu komponentów. Co więcej, w przypadku pojawienia się zakłócenia zewnętrznego adaptacyjne rozmieszczanie umożliwia redukcję czasu wykonania.

Zdaniem autora zastosowanie rozmieszczania adaptacyjnego w heterogenicznych środowiskach rozproszonych jest jednym z kluczowych elementów, pozwalających zrealizować efektywną adaptację aplikacji. Stanowi to silną motywację do dalszych badań w tym kierunku, m.in. wsparcia dla rozmieszczania w systemach zwirtualizowanych, udoskonalenia algorytmu planowania FDDP czy rozwiązania planowania w wymiarach czasowym i semantycznym.

Literatura

[1] P. Antoniewski, Ł. Cygan, J. Cała, and K. Zieliński. Extension of a CCM environment with and adaptive planning mechanism. In *CISSE 2006*, 2006.

[2] J. Cała, Ł. Kubica, W. Wiśniowski, and K. Zieliński. Adaptation of CCM applications based on lightweight OS virtualization. In *FeBID 2007 — Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, May 2007.

[3] J. Cała and K. Zieliński. Influence of virtualization on process of grid application deployment — CCM case study. In *Cracow Grid Workshop 2006*, 2006.

[4] A. Carzaniga, A. Fuggetta, R.S. Hall, A. van der Hoek D. Heimbigner, and A.L. Wolf. A characterization framework for software deployment technologies. Technical Report CU-CS-857-98, University of Colorado, Department of Computer Science, April 1998. Air Force Material Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Number F30602-94-C-0253.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. *Approximation Algorithms for NP-Hard Problems*, chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[6] G. Coulson. What is reflective middleware? *IEEE Distributed Systems Online*, 2000.

[7] A. Dearle. Software deployment, past, present and future. In *International Conference on Software Engineering*, pages 269–284. IEEE Computer Society, 2007.

[8] Distributed Management Task Force, Inc. *CIM Schema*, 2008. Version 2.18.1.

[9] C. Perkins et al. *IP Mobility Support for IPv4*. The Internet Engineering Task Force, August 2002. Request for Comments: 3344.

[10] A. Fuggeta, G.P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, ?(5):342–361, May 1998.

[11] A. Gokhale, K. Balasubramanian, and J. Balasubramanian et al. Model driven middleware: A new paradigm for deploying and provisioning distributed real-time and embedded applications. *Elsevier Journal of Science of Computer Programming: Special Issue on Model Driven Architecture*, 2004.

[12] IBM. An architectural blueprint for autonomic computing, June 2005.

[13] E. Kotsovinos. *Global Public Computing*. PhD thesis, University of Cambridge; Computer Laboratory, JJ Thomson Avenue, Cambridge, UK, January 2005.

[14] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16:1293–1306, 1990.

[15] P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. The Internet Engineering Task Force, July 2005. Request for Comments: 4122.

[16] P. Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155, 1987.

[17] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile application-aware adaptation for mobility. In *SOSP '97: Proceedings*

of the sixteenth ACM symposium on Operating systems principles, pages 276–287, New York, NY, USA, 1997. ACM.

[18] Object Management Group, Inc. *Deployment and Configuration of Component-based Distributed Applications Specification*, April 2006. Version 4.0.

[19] Object Management Group, Inc. *Quality of Service for CORBA Components, Beta 2*, September 2007.

[20] Object Management Group, Inc. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1; Part 3: CORBA Component Model*, January 2008.

[21] Sun Microsystems, Inc. *JDK 6 Documentation*, 2006.

[22] C. Szyperski. *Oprogramowanie komponentowe; obiekty to za mało*. WNT, Warszawa, Polska, 2001.

[23] V. Talwar, D. Milojicic, Q. Wu, C. Pu, W. Yan, and G. Jung. Approaches for service deployment. *IEEE Internet Computing*, 9(2):70–80, 2005.

[24] R. Tamassia. Handbook of graph drawing and visualization. Online, not finished yet, 2008.