AUTOMATYKA • 2008 • Tom 12 • Zeszyt 3

Grzegorz Ciesielski*, Andrzej Albrecht*, Rafał Wojciechowski*

Projekt środowiska rozproszonego do badania systemów nieliniowych z wykorzystaniem technologii Java RMI

1. Wprowadzenie

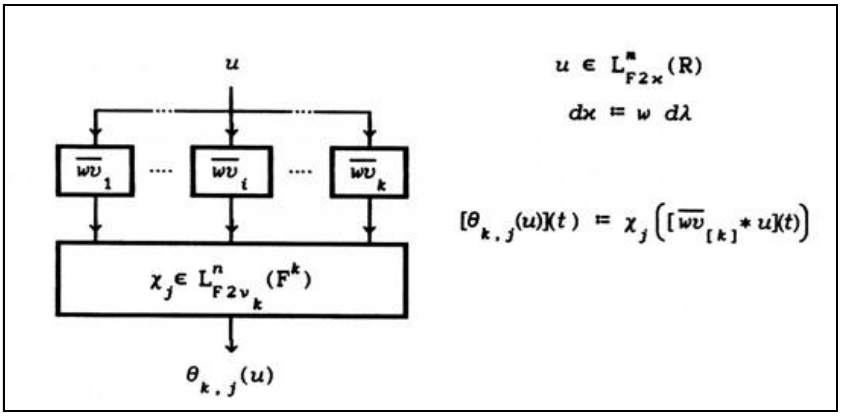
Symulacja komputerowa odgrywa współcześnie ważną rolę w rozwiązywaniu szeregu złożonych problemów technicznych znajdujących miejsce w niemalże wszystkich dziedzinach przemysłu, jak np.: w metrologii, automatyce, telekomunikacji, projektowaniu układów wielkiej skali integracji, bioinżynierii czy chemii. Wielokrotnie stanowi pierwszy krok w opracowywaniu nowych technologii. Złożoność rozpatrywanych problemów i mnogość przypadków testowych pociągają za sobą dużą ilość obliczeń, co z kolei wymaga zastosowania odpowiednich systemów komputerowych charakteryzujących się dużą mocą obliczeniową. W rozwiązywaniu współczesnych problemów naukowych coraz większą popularność zyskuje przetwarzanie rozproszone oferujące stosunkowo łatwą możliwość zwiększenia całkowitej mocy obliczeniowej systemów komputerowych. Współczesne wieloprocesorowe rozwiązania sprzętowe oraz ogólna dostępność sieci komputerowych stwarzają możliwość efektywnej realizacji zadań skomplikowanych obliczeniowo w drodze zrównoleglenia obliczeń.

2. Przetwarzanie rozproszone w zastosowaniu do badań systemów nieliniowych

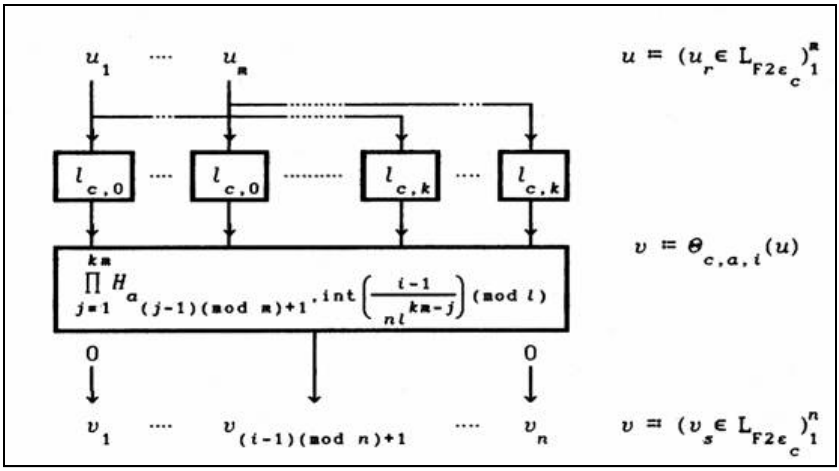
Z funkcjonalnej teorii systemów nieliniowych (FTSN), opublikowanej przez G. Ciesielskiego [1] w 1994 r., wynika struktura *C*-operatora $\theta_{k,j}$, którą przedstawiono na rysunku 1, a omówiono szczegółowo w [1]. Teoria ta jest uogólnieniem teorii systemów nieliniowych V. Volterry i N. Wienera.

Z tej ogólnej struktury *C*-Operatora $\theta_{k,j}$ wynika, jak to zostało pokazane na rysunku 2, struktura operatora Wienera $\theta_{c,a,k.i}$.

* Katedra Informatyki Stosowanej, Politechnika Łódzka



Rys. 1. Struktura C-operatora $\theta_{k,j}$ ([1], s. 61)



Rys. 2. Struktura operatora Wienera $\theta_{c,a,k,i}$ ([1], s. 91)

Niech dla przykładu *m* będzie liczbą wejść, *n* liczbą wyjść ortonormalnych operatorów Wienera. Załóżmy ponadto, że do ich konstrukcji użyto *k* pierwszych funkcji Legendre'a $(l_{c,i\text{-}1})^{k}_{1}$ oraz *l* pierwszych wielomianów Hermite'a $(H_{a,j\text{-}1})^{l}_{1}$. Wówczas układ operatorów Wienera ma postać:

$$l_{c[k]} := \left(l_{c,i-1}\right)_1^k,$$

$$\forall t \in R \quad \left(\lambda_{c[km]}(t) := \left(\lambda_{c,i}(t)\right)_1^{km} := \mathrm{vect}\left(l_{c[k]}(t) \otimes I_{[m,n]}\right)\right),$$

$$\left(H_{a_j}[l] \coloneqq \left(H_{a_j,i-1} \right)_1^l \right)_1^m , a \coloneqq (a_i)_1^m ,$$

$$\forall u \in L^m_{R2\varepsilon_c}(R) \quad \forall t \in R \quad \left(\left[\theta_{c,a\left[nl^{km}\right]}(u) \right](t) \coloneqq \left(\left[\theta_{c,a,i}(u) \right](t) \right)_1^{nl^{km}} \right) \coloneqq$$

$$\coloneqq \operatorname{vect}\left(\left(\bigotimes_{j=1}^{km} H_{a_{(j-1)(\bmod m)+1}[l]} \left(\left[\lambda_{c[km]} * u \right](t) \right) \right) \otimes I_{[n,n]} \right).$$

Zatem liczba tych złożonych ortonormalnych operatorów Wienera jest równa $\Omega = nl^{km}$ ([1], s 90). Tą złożoność obliczeniową, tzn. liczbę ortonormalnych operatorów Wienera Ω dla $n = 1$ oraz $k = l$ najlepiej ilustruje tabela 1.

Tabela 1
Liczba ortonormalnych operatorów Wienera Ω dla $n = 1$ oraz $k = l$

	l				
m	2	3	4	5	6
1	4	27	256	3125	46656
2	16	729	65536	9765625	2176782336
3	64	19683	16777216	3,051757813e10	1,015599567e14

Zauważmy, że tabela 1 zawiera wymiary przestrzeni span $\theta_{c,a,k/\Omega/}$, czyli wartości dim span $\theta_{c,a,k/\Omega/}$. Przestrzeń ta jest powłoką liniową układu $\theta_{c,a,k/\Omega/}$, czyli jest utworzona na zbiorze wszystkich liniowych kombinacji elementów układu $\theta_{c,a,k/\Omega/}$. Wydaje się, że warianty zaznaczone w tabeli kolorem jasnoszarym są w zasięgu mocy obliczeniowej współczesnych czterordzeniowych komputerów klasy PC z zegarem 3 GHz, których moc obliczeniową można oszacować na poziomie ok. 25 GFPS, warianty zaznaczone kolorem szarym prawdopodobnie mogą być obliczone na superkomputerach, np. na Galerze z CI TASK[1)], a warianty zaznaczone kolorem ciemnoszarym są z pewnością nieliczalne przy współczesnym poziomie rozwoju informatyki.

Na podkreślenie zasługuje tu bardzo ciekawy fakt, że tabela 1 zawiera wymiary przestrzeni span $\theta_{c,a,k/\Omega/}$, w której poszukujemy rozwiązań, tzn. modeli lub korektorów syste-

---

1) Klaster obliczeniowy wyposażony w 1344 procesory Xeon Quad-Core pracujące w architekturze SMP z siecią InfiniBand, oferujący teoretyczną mocą obliczeniową rzędu 50 TFPS. Klaster znajduje się w Centrum Informatycznym Trójmiejskiej Akademickiej Sieci Komputerowej (CI TASK).

mów nieliniowych. Współczesna matematyka mówi, że wymiaru przestrzeni, a więc i bazy tej przestrzeni, nie da się ominąć przy obliczeniach. Istnieje więc bariera obliczeniowa modelowania i korekcji, stąd celowość poszukiwania rozwiązań rozproszonych umożliwiających zwiększenie mocy obliczeniowej.

3. Java RMI [2, 5]

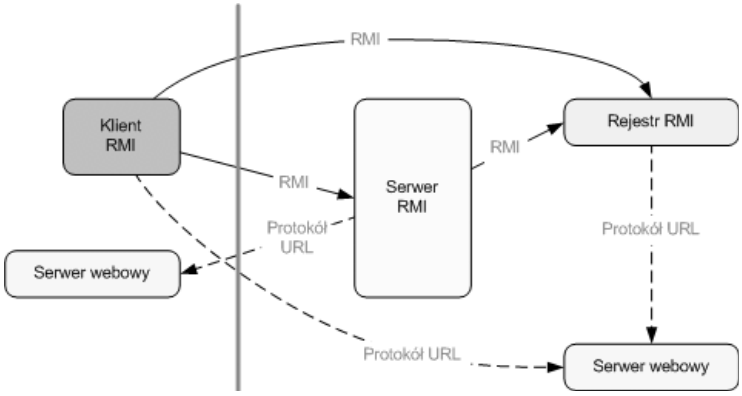
RMI (*Remote Method Invocation*) jest technologią pozwalającą na zorganizowanie komunikacji między obiektami oprogramowanymi w języku Java, działającymi na odrębnych, rozproszonych w środowisku sieciowym, maszynach wirtualnych, w sposób zbliżony do tego, jaki stosuje się dla obiektów działających lokalnie – za pomocą wywołań metod, ukrywając przed programistą większość niskopoziomowych szczegółów związanych z obsługą protokołów sieciowych. Takie podejście, obok wygody i łatwości implementacji, daje możliwość zaprojektowania systemu rozproszonego w sposób obiektowy, tak dalece, jak to tylko możliwe.

Aplikacje RMI składają się najczęściej z części serwerowej oraz klienckiej. Serwer tworzy zdalne obiekty, umożliwia korzystanie z nich poprzez udostępnienie referencji do nich i oczekuje zgłoszeń klienta, aby wywołać na nich określone metody. Klient otrzymuje referencje do określonych obiektów serwera, do metod których może się odwołać. RMI wprowadza mechanizm komunikacji między serwerem a klientem oraz normuje przepływ informacji w obydwu kierunkach. Ten rodzaj aplikacji określany jest często mianem aplikacji pracującej na obiektach rozproszonych.

Z uwagi na rozproszony charakter obiektów, konieczne jest spełnienie określonych wymagań dotyczących:

- lokalizacji zdalnych obiektów – aplikacje mogą wykorzystywać różne mechanizmy pozyskania referencji do zdalnych obiektów, przykładowo, aplikacja może wykorzystać prosty serwis nazewniczy – rejestr RMI – w celu zarejestrowania obiektów;
- komunikacji ze zdalnymi obiektami – komunikacja pomiędzy obiektami jest obsługiwana przez mechanizmy RMI, dla projektanta, komunikacja zdalna wygląda analogicznie jak zwykłe wywołanie metod Javy;
- ładowania definicji klas dla obiektów, których wywołania dotyczą – ponieważ RMI pozwala na przekazywanie dwukierunkowe obiektów, wprowadza mechanizmy zarówno ładujące definicje klas, jak również przenoszące dane obiektów.

Schemat pokazany na rysunku 3 ilustruje działanie rozproszonej aplikacji wykorzystującej rejestr RMI w celu pobrania referencji do zdalnego obiektu. Serwer odwołuje się do rejestru w celu dowiązania nazwy do zdalnego obiektu. Klient poszukuje obiektu, identyfikując go po nazwie w rejestrze serwera, a następnie wywołuje na nim określone metody. Na rysunku 3 pokazano również wykorzystanie istniejącego serwera HTTP w celu ładowania definicji klas.



Rys. 3. Działanie rozproszonej aplikacji wykorzystującej rejestr RMI [5]

4. Projekt środowiska rozproszonego do badania systemów nieliniowych

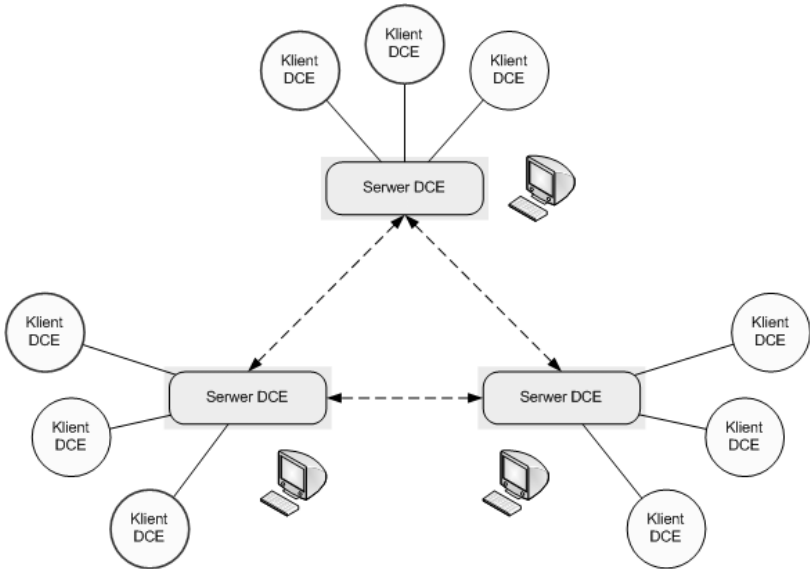
4.1. Koncepcja środowiska rozproszonego do badania systemów nieliniowych

Koncepcja środowiska rozproszonego (rys. 4) do badania systemów nieliniowych zakłada istnienie dwóch funkcjonalnych grup programów pełniących role aktorów w zaprojektowanym systemie – są to serwery środowiskowe, uruchamiane w dowolnych węzłach sieci oraz moduły klienckie, skojarzone z konkretnymi serwerami, pełniące funkcje określone przez użytkownika. Rola serwerów w powyższym schemacie sprowadza się do zarządzania modułami klienckimi – uruchomienia, zatrzymania oraz ustawienia ich własności, zgodnie ze zgłoszeniami aplikacji zarządzającej, inicjującej powyższe akcje. Moduły klienckie dokonują wymiany danych na zasadach publikacji, przy czym nie ma znaczenia tutaj fakt uruchomienia modułów na tym samym serwerze (lokalnie, na tym samym węźle), czy też na innych serwerach (operujących na zdalnych węzłach). Ogólną koncepcję środowiska rozproszonego zilustrowano na rysunku 4.

Wystosowano następujące założenia oraz wymagania względem rozproszonego środowiska do badania systemów nieliniowych:

- implementacja strony serwerowej i klienckiej systemu rozproszonego z wykorzystaniem języka Java;
- zapewnienie możliwości wymiany danych pomiędzy dowolnymi programami klienckimi pracującymi na jednym lub wielu węzłach sieciowych za pomocą protokołu komunikacyjnego niezależnego od technologii wykonania sieci i platformy systemowej (z wykorzystaniem Java RMI);
- wystosowanie odpowiedniego API dla aplikacji klienckich umożliwiającego dokonywanie podstawowych operacji na zmiennych i wykonanie predefiniowanych domyślnych akcji modułów (callbacków);

– ujednolicenie klasy prototypów danych umożliwiającej przesyłanie dowolnych informacji w sieci;
– opracowanie odpowiedniego języka skryptowego umożliwiające zestawienie odpowiedniej konfiguracji dla środowiska rozproszonego;
– zapewnienie możliwości monitoringu i zarządzania aplikacjami serwera (z wykorzystaniem Java JMX).



Rys. 4. Ogólna koncepcja środowiska rozproszonego

4.2. Protokół komunikacyjny

Architektura systemu zakłada współpracę pomiędzy modułami klienckimi zarządzanymi przez serwery w środowisku rozproszonym. Protokół komunikacyjny definiuje podstawową funkcjonalność na poziomie aplikacja zarządzająca – serwer środowiskowy – moduł klienta, opisując niezbędne operacje dla uruchomienia, zatrzymania oraz ustawienia własności modułów w nim zarejestrowanych. Serwery działają w trybie zgłoszeniowym, oczekują odpowiednich wywołań poprzez Java RMI, definiując podstawowe operacje możliwe do wykonania, zgodnie z tabelą 2. Rysunek 5 określa przypadki użycia systemu rozproszonego, na rysunku 6 przedstawiono natomiast schemat akcji ustawienia zmiennej lokalnego oraz zdalnego modułu.

4.3. Moduły klienckie

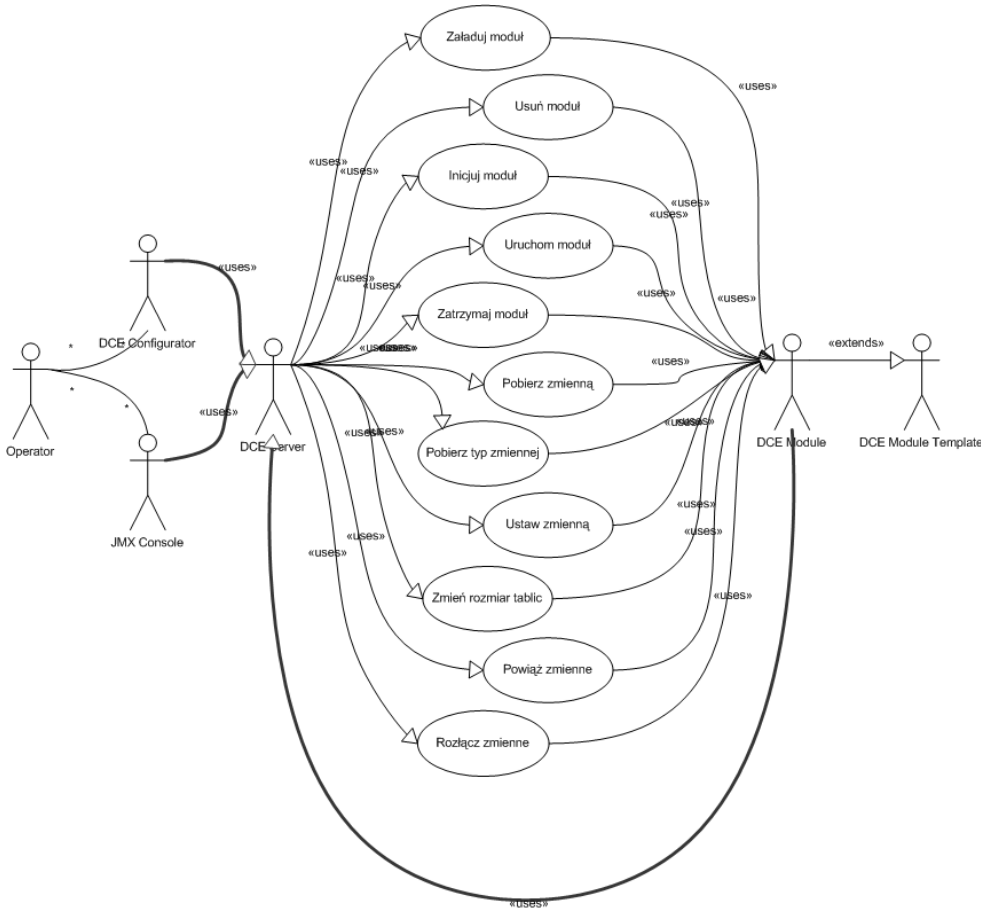
Środowisko rozproszone do badań systemów nieliniowych definiuje odpowiednie API klienta definiujące podstawowe reakcje modułu na ustawienie i pobranie jego zmiennych, jak również domyślną akcję wykonywaną cyklicznie na pobudzeniu wejścia modułu. Za-

łożeniem projektowym była organizacja zmiennych modułów w taki sposób, aby odpowiadały funkcjonalnie grupie zmiennych wejściowych, wyjściowych oraz własności definiujących specyficzne zmienne modułu (przykładowo wartość wzmocnienia dla modułów wzmacniaków). Poszczególne grupy zmiennych przechowywane są w dynamicznych tablicach i reprezentowane są przez typ *java.lang.Object*. Wymiana danych między modułami możliwa jest w sposób bezpośredni (w przypadku, jeśli moduły pracują na tym samym serwerze) lub też pośredni (w przypadku, jeśli moduły zlokalizowane są w zdalnych węzłach sieci).

Tabela 2
Podstawowe operacje serwera środowiskowego

Interfejs RMI serwera	Opis
loadModule	Załadowanie wątku modułu
killModule	Usunięcie wątku modułu i wszystkich powiązanych zasobów
initModule	Inicjalizacja modułu
startModule	Uruchomienie modułu
stopModule	Zatrzymanie modułu
sizeModuleVar	Zmiana rozmiaru tablicy zmiennych modułu
setModuleVar	Ustawienie zmiennej modułu
getModuleVar	Pobranie zmiennej modułu
getModuleVarType	Pobranie typu zmiennej modułu
linkModuleVar	Powiązanie zmiennych dwóch modułów
unlinkModuleVar	Usunięcie powiązania zmiennych dwóch modułów

Funkcjonowanie obydwu metod wymaga zdefiniowania w każdym z modułów tablicy powiązań między zmienną wejściową modułu, która będzie przechowywała bieżącą wartość zmiennej modułu publikującego oraz zmienną wyjściową tego modułu. W przypadku komunikacji bezpośredniej moduł przechowuje deskryptor odpowiedniego procesu (otrzymywany z serwera w momencie konfiguracji powiązania zmiennych) i odwołuje się do zmiennych modułu subskrybującego poprzez publiczne metody `get` oraz `set` powiązane z określonymi grupami zmiennych. Komunikacja pośrednia natomiast wykonywana jest poprzez wywołanie metody ustawienia zmiennej w module docelowym za pośrednictwem serwera, na którym moduł docelowy jest zarejestrowany, z wykorzystaniem Java RMI. Wszelkie parametry modułu docelowego (adres serwera, nazwa modułu oraz identyfikator zmiennej) są przekazywane również na etapie konfiguracji środowiska. Domyślna akcja modułu wykonywana jest w momencie ustawienia wszystkich zmiennych wejściowych modułu – zakłada się więc, że moduł musi posiadać przynajmniej jedną zmienną wejściową (schemat wywołania domyślnej akcji wątku klienckiego zaprezentowano na rys. 7).



Rys. 5. Przypadki użycia środowiska rozproszonego
(pogrubioną linią oznaczono wywołania poprzez Java RMI)

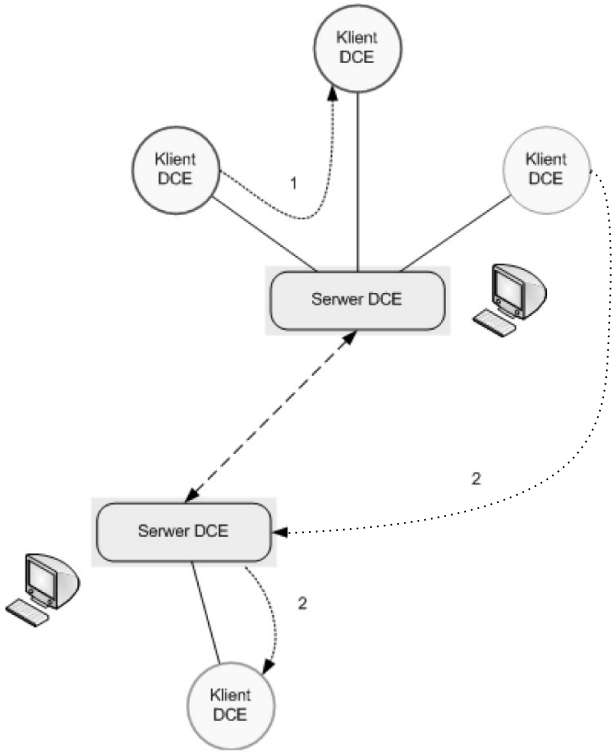
4.4. Monitoring i zarządzanie rozproszonym środowiskiem do badania systemów nieliniowych

W celu zautomatyzowania i ułatwienia pracy użytkownikowi, zaprojektowano odpowiedni język skryptowy pozwalający na wykonanie podstawowych operacji serwera, zgodnie z tabelą 2. Interpreter języka skryptowego dokonuje wywołań odpowiednich funkcji interfejsu serwera poprzez Java RMI, mając możliwość konfiguracji modułów. Poniżej zaprezentowano składnię języka skryptowego.

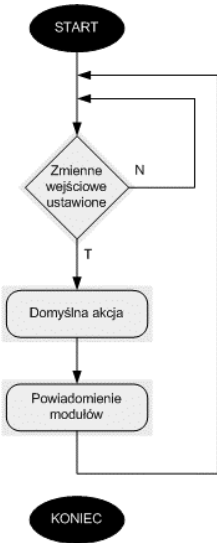
```
load    [nazwa modułu] [adres serwera] [klasa modułu]
kill    [nazwa modułu] [adres serwera]
size    [nazwa modułu] [adres serwera] [liczba zmiennych wejściowych]
        [liczba zmiennych wyjściowych] [liczba zmiennych własności]
```

```
set     [nazwa modułu] [adres serwera] [identyfikator tablicy zmiennych]
        [pozycja zmiennej] [typ zmiennej = String | Integer | Double]
        [wartość zmiennej]
link    [nazwa modułu wyjściowego] [adres serwera wyjściowego]
        [pozycja zmiennej wyjściowej] [nazwa modułu wejściowego]
        [adres serwera wejściowego] [pozycja zmiennej wejściowej]
unlink  [nazwa modułu wyjściowego] [adres serwera wyjściowego]
        [pozycja zmiennej wyjściowej] [nazwa modułu wejściowego]
        [adres serwera wejściowego] [pozycja zmiennej wejściowej]
init    [nazwa modułu] [adres serwera]
start   [nazwa modułu] [adres serwera]
stop    [nazwa modułu] [adres serwera]
```

Dodatkowo, moduły serwera wyposażono w możliwość ręcznego wywołania ww. metod oraz wyświetlenia statystyk i obciążenia serwera poprzez konsolę Java JMX.



Rys. 6. Schemat akcji ustawienia zmiennej zdalnego modułu (1 – sposób bezpośredni poprzez metody getVar oraz setVar na określonym module klienta lokalnego, 2 – sposób pośredni z wykorzystaniem Java RMI, wywołanie setVar na zdalnym module klienta poprzez serwer DCE)



Rys. 7. Schemat wywołania akcji *execute* wątku modułu klienckiego

5. Podsumowanie

W artykule zaprezentowano projekt rozproszonego środowiska ogólnego przeznaczenia, na którego bazie możliwe jest zbudowanie systemu do badania i modelowania systemów nieliniowych. Proponowane rozwiązanie stanowi implementację znanego mechanizmu komunikacji z zastosowaniem ogólnie dostępnego narzędzia, jakim jest Java RMI. Dostępne rozwiązania (np. Cogent Cascade DataHub [4], środowisko IMC dla systemu QNX [3], protokół OPC, DCOM, …) stanowią często mechanizmy ściśle dedykowane, co zawęża obszar ich zastosowania oraz możliwości dalszej rozbudowy. W artykule zaprezentowano specyfikację wymagań oraz szczegóły implementacyjne środowiska rozproszonego z wykorzystaniem technologii Java RMI. Proponowane rozwiązanie stanowi podstawę dla rozwijanego systemu modelowania i korekcji systemów nieliniowych.

Literatura

[1] Ciesielski G., *Modelowanie i korekcja wielowymiarowych stacjonarnych systemów pomiarowych za pomocą operatorów nieliniowych*. Zeszyty Naukowe Politechniki Łódzkiej nr 699, Łódź, 1994 (rozprawa habilitacyjna).
[2] Eckel B., *Thinking in Java*. Prentice-Hall, 2002.
[3] Wojciechowski R., Kołodziejski H., Albrecht A., *Implementacja serwera danych w systemie czasu rzeczywistego QNX*. XII Konferencja Sieci i Systemy, materiały konferencyjne, Łódź, 2004 393–400.
[4] Strona WWW: *Cogent Real-Time Systems*. http://www.cogent.ca, 2006.
[5] Strona WWW: *An Overview of RMI Applications*. http://java.sun.com/docs/books/tutorial/rmi, 2008.