DOI: 10.24427/978-83-66391-28-4_10

Symulator systemów klasy SOA

Adrian P. WOŹNIAK[*]

1. Architektura zorientowana na usługi

Architektura zorientowana na usługi (ang. *Service Oriented Architecture*, SOA) to sposób wytwarzania oprogramowania mający na celu w optymalny sposób wspierać organizację. Składa się na to wiele cech, z których najważniejszymi są czas i koszty wprowadzania zmian. Dzięki stosowaniu usług można zapewnić niezależność w rozwoju poszczególnych systemów i rozwijać je równolegle. Pozwala to na wprowadzanie zmian w organizacji w krótkim czasie, co staje się coraz bardziej istotne we współczesnym, szybko zmieniającym się świecie. Od 15 lat podejście to zyskuje na popularności i jest coraz szerzej wdrażane w przedsiębiorstwach. Współcześnie koncepcja ta jest rozwijana z wykorzystaniem coraz to nowszych technologii. Skutkuje to pojawieniem się nowych koncepcji rozszerzających SOA, takich jak np. tzw. mikrousługi (ang. *microservices*), gdzie nowe systemy tworzone są z niewielkich komponentów odpowiedzialnych za wydzielony i dobrze wyspecyfikowany obszar biznesowy.

Centralnym pojęciem w SOA jest usługa, która jest niezależną funkcją systemu, dającą wartość z punktu widzenia procesu biznesowego. Każda usługa może być więc krokiem w procesie biznesowym. Najczęściej są one udostępniane poprzez sieć w formie tzw. Web Services. Usługi dostarczane są przez komponenty, czyli oprogramowanie, które jest względem siebie niezależne. Oznacza to, że każdy komponent może zostać wytworzony w innej technologii i działać na innym serwerze, w innym środowisku uruchomieniowym. Jest to poprawne z punktu widzenia SOA, dopóki usługi udostępniane są w uzgodnionej wcześniej formie. Elastyczność SOA pozwala na to, aby z jednej strony wytwarzać systemy w formie np. tzw. mikrousług, a z drugiej strony modyfikować istniejące stare systemy tak, aby udostępniać ich funkcje w formie usług. Powoduje to rosnącą liczbę stosowanych technologii, środowisk uruchomieniowych (rozumianych jako oprogramowanie potrzebne do uruchomienia komponentu) i serwerów. W celu realizacji procesu biznesowego potrzebne jest wywołanie wielu usług udostępnianych przez liczne komponenty. Oznacza to, że realizację procesu biznesowego angażowanych może być wiele serwerów komunikujących się poprzez sieć.

[*] Wojskowa Akademia Techniczna

Istotne jest zatem takie wdrożenie komponentów na serwerach, aby proces biznesowy realizowany był jak najlepiej z punktu widzenia organizacji. Poprzez „jak najlepiej" rozumiemy tutaj optymalizację według czterech kryteriów:

- czasu i wariancji procesu biznesowego,
- zużycia procesora i pamięci RAM na rzecz usług względem wszystkich zarezerwowanych zasobów.

Zastosowanie pierwszych dwóch kryteriów wydaje się być intuicyjne – z punktu widzenia organizacji wartościowe jest, aby proces biznesowy realizowany był w czasie jak najkrótszym i żeby czas ten był jak najbardziej przewidywalny. Minimalizacja według pozostałych dwóch kryteriów ma na celu jak najlepsze wykorzystanie zasobów organizacji. Zastosowanie wielu serwerów o dużej mocy obliczeniowej może dać lepsze wyniki pod względem pierwszych dwóch kryteriów, ale niekoniecznie musi być optymalne dla firmy, ponieważ może się okazać, że serwery te będą w niewielkim stopniu wykorzystywane, a takie rozwiązanie byłoby nieekonomiczne.

Podjęte próby optymalizacji w obszarze SOA skupiały się głównie na procesie tzw. Service Selection, a więc na algorytmie wyboru usługi w przypadku, gdy wiele instancji komponentu udostępnia tę samą usługę i należy wybrać ten, który ją zrealizuje. Wykorzystano do tego wiele koncepcji i algorytmów. W [1], [2], [3], [4] i [5] można znaleźć przykłady metod Service Selection opartych o algorytm genetyczny. Metoda polegająca na przeszukiwaniu drzew binarnych została opisana w [6]. Znacząco mniej popularnym zagadnieniem jest optymalizacja przydziału komponentów do serwerów. Tekst [7] przedstawia metodę optymalizującą przydział pod względem kosztowym przy ograniczeniach, jakie powoduje SLA (ang. *Service Level Agreement*). W artykule [8] można znaleźć metodę optymalizującą dostępność usług. Natomiast w [9] połączone są aspekty optymalizacji Service Selection (optymalizacja krótkoterminowa) z długoterminowo optymalnym przydziałem zasobów do serwerów. W literaturze można także znaleźć metody harmonogramowania usług. Definiowane jest ono jako wybór kolejności realizacji usług w kolejce komponentu. FIFO jest najczęściej stosowanym, choć niekoniecznie optymalnym, sposobem organizacji kolejki. Propozycję metody harmonogramowania usług można znaleźć w [10]. Jej działanie polega na znalezieniu usług będących na tzw. ścieżce krytycznej procesu i priorytetyzowaniu realizacji właśnie ich. W [11] natomiast można znaleźć bardziej rozbudowaną metodę, która optymalizuje kolejność na dwóch poziomach:

- globalnym, którym jest maksymalizacja prawdopodobieństwa spełnienia ograniczeń jakości usług (QoS),
- lokalnym, którym jest realizacja jednej z czterech polityk: najpierw usługi o największej wartości dodanej, najpierw usługi o najkrótszym wymaganym czasie realizacji (wynikającym z QoS), najpierw usługi o największej proporcji poprzednich dwóch kryteriów, najpierw usługi wybrane metodą Lawlera.

Jak dotąd nie zostały opublikowane metody optymalizujące przydział komponentów do serwerów w architekturze SOA, uwzględniające jednocześnie opisane wcześniej złożone działanie takich systemów, niezawodność serwerów oraz algorytmy wyboru usług i kolejności ich realizacji w kolejkach. Do przygotowania takiego optymalizatora proponowany jest algorytm genetyczny, którego celem jest znalezienie optymalnego przydziału komponentów do serwerów przy zadanych algorytmach Service Selection oraz wybór kolejności realizacji usług w kolejkach. Najtrudniejszym elementem takiego optymalizatora jest ocena rozwiązania. Do oceny wykorzystany został symulator, którego celem jest pomiar wymienionych wcześniej czterech kryteriów rozwiązania.

2. Środowisko symulacyjne

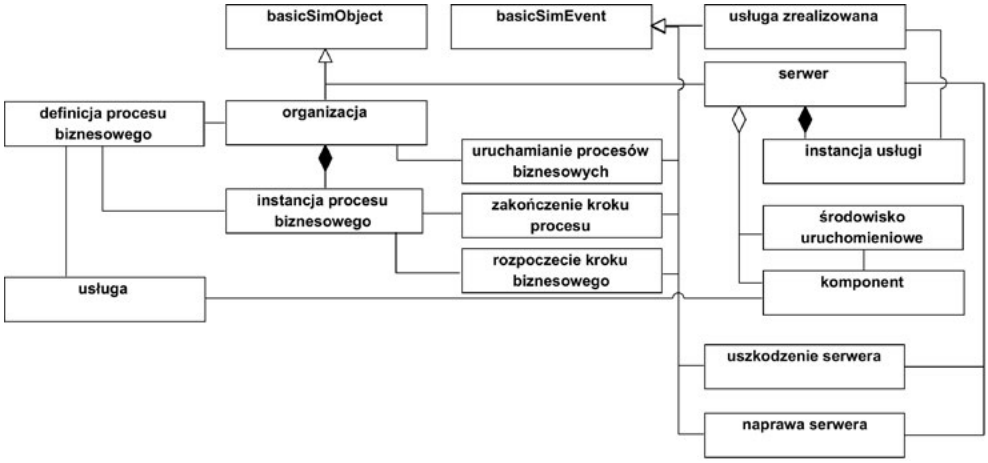
Symulator systemów klasy SOA został zaimplementowany z wykorzystaniem dwóch środowisk symulacyjnych: DESKit oraz DISSim. Obydwa środowiska zostały napisane w języku JAVA oraz wspierają symulację dyskretną zdarzeniową, która została tu zastosowana. Występuje jednak między nimi znacząca różnica koncepcyjna. W środowisku DESKit wykorzystuje się dwie najważniejsze klasy: BasicSimObject oraz BasicSimActivity. Pierwsza z nich reprezentuje obiekty symulacyjne, takie jak np. serwer czy organizacja, natomiast druga działania tych obiektów – np.: instancję procesu biznesowego i realizację usług przez serwer. Najważniejszą cechą tego środowiska jest to, że każda aktywność jest reprezentowana jako osobny wątek. Symulator oparty o to środowisko wznawia i zatrzymuje wątki zgodnie z kolejnością zapisaną w tzw. Pending List. W środowisku DISSim natomiast najważniejszymi klasami są BasicSimObject oraz BasicSimEvent. Pierwsza z nich, analogicznie do DESKit, reprezentuje obiekty symulacyjne, a druga odnosi się do zdarzeń w świecie symulacyjnym. Takimi zdarzeniami są np.: rozpoczęcie lub zakończenie kroku w procesie, realizacja usługi w serwerze czy uszkodzenie serwera. Podejście zdarzeniowe to najważniejsza różnica koncepcyjna względem DESKit. Program w tym środowisku jest jednowątkowy i realizuje kod zawarty w zdarzeniach w kolejności od najwcześniejszego według zapisów zawartych w kalendarzu. Gdy czas symulacji osiągnie czas zdarzenia symulacyjnego, realizowana jest zawartość metody stateChange.

Na wysokim poziomie koncepcyjnym łatwiej jest projektować w środowisku DESKit, ponieważ bardziej odpowiada on intuicyjnemu rozumieniu świata symulacyjnego (obiekty i ich działania). Z kolei na niskim poziomie koncepcyjnym (implementacyjnym) łatwiej projektować w środowisku DISSim, ponieważ nie trzeba mieć na uwadze wielowątkowości, synchronizacji itd. Ponadto jednowątkowy program łatwiej jest debugować. Rozstrzygającym czynnikiem powodującym porzucenie środowiska DESKit na rzecz DISSim jest wydajność. W testach, w zależności od liczby instancji procesu (w DESKit – równoległych wątków), różnica w wydajności między rozwiązaniami była od kilkukrotnej (dla kilkuset procesów) do czterystukrotnej (dla 40 tys.

procesów biznesowych). Relacja ta jest zatem nieliniowa. Wynika to z potrzeby przełączania wątków, która bardziej obciąża zasoby. Ponadto przetwarzanie wielowątkowe, w tym wypadku, nie niesie za sobą korzyści, ponieważ w jednej chwili zawsze przetwarzany jest jeden wątek, a pozostałe czekają na swoją kolej. Wykorzystanie wydajnego środowiska jest bardziej wskazane, ponieważ zastosowany w optymalizatorze algorytm genetyczny wymaga oceny setek genów w setkach iteracji.

3. Model klas symulatora

Jak zaznaczono wcześniej, w środowisku DISSim najważniejszymi klasami są BasicSimObject oraz BasicSimEvent. Są to klasy abstrakcyjne, z których dziedziczyć powinny klasy biorące udział w symulacji. W przypadku symulatora systemów klasy SOA obiektami symulacyjnymi są organizacja oraz serwery. Organizacja uruchamia i zawiera w sobie instancje procesów biznesowych, zgodnie z definicją procesu. Proces biznesowy został zapisany jako graf, w którym każdy krok jest usługą. Usługi w procesie połączone są między sobą łukami opisanymi prawdopodobieństwem wyboru każdej ze ścieżek (co odzwierciedla działanie bramek w BPMN). Usługi opisane są mocą procesora i RAM-u, jakiego potrzebują do swojego działania oraz ilością danych, która powinna być przesłana przez sieć, aby usługę zrealizować. Usługi przypisane są do komponentów, które je realizują. Komponenty natomiast zawierają listę środowisk uruchomieniowych, na których mogą działać.



RYS. 1. Diagram klas symulatora systemów SOA
FIG. 1. Class diagram of SOA systems simulator

ŹRÓDŁO: opracowanie własne.
SOURCE: own elaboration.

Komponenty i środowiska są uruchamiane na serwerze, który jest obiektem symulacyjnym i opisane są mocą procesora oraz pamięcią RAM, jaka jest niezbędna do ich działania. Z kolei serwery są opisane posiadanymi zasobami niezbędnymi do uruchomienia komponentów i środowisk uruchomieniowych oraz do realizacji usług. Każdy obiekt klasy „serwer" zawiera także zmienne losowe oznaczające czas do uszkodzenia i naprawy. Ponadto serwery są opisane macierzą przepustowości sieci między nimi. Obiekty symulacyjne (serwery i organizacja) mają przypisane zdarzenia będące klasami dziedziczącymi z BasicSimEvent przedstawione na rysunku 1.

Zdarzenia uszkodzenia tworzone są po uruchomieniu symulatora w czasie wylosowanym zgodnie ze zmienną przypisaną do serwera. Gdy pojawi się zdarzenie uszkodzenia serwera, jego status zmieniany jest na niesprawny i generowane jest zdarzenie „naprawa serwera" o losowym czasie od czasu uszkodzenia. Zdarzenie naprawy generuje zdarzenie uszkodzenia itd.

4. Sposób działania

4.1. Uruchamianie procesów biznesowych

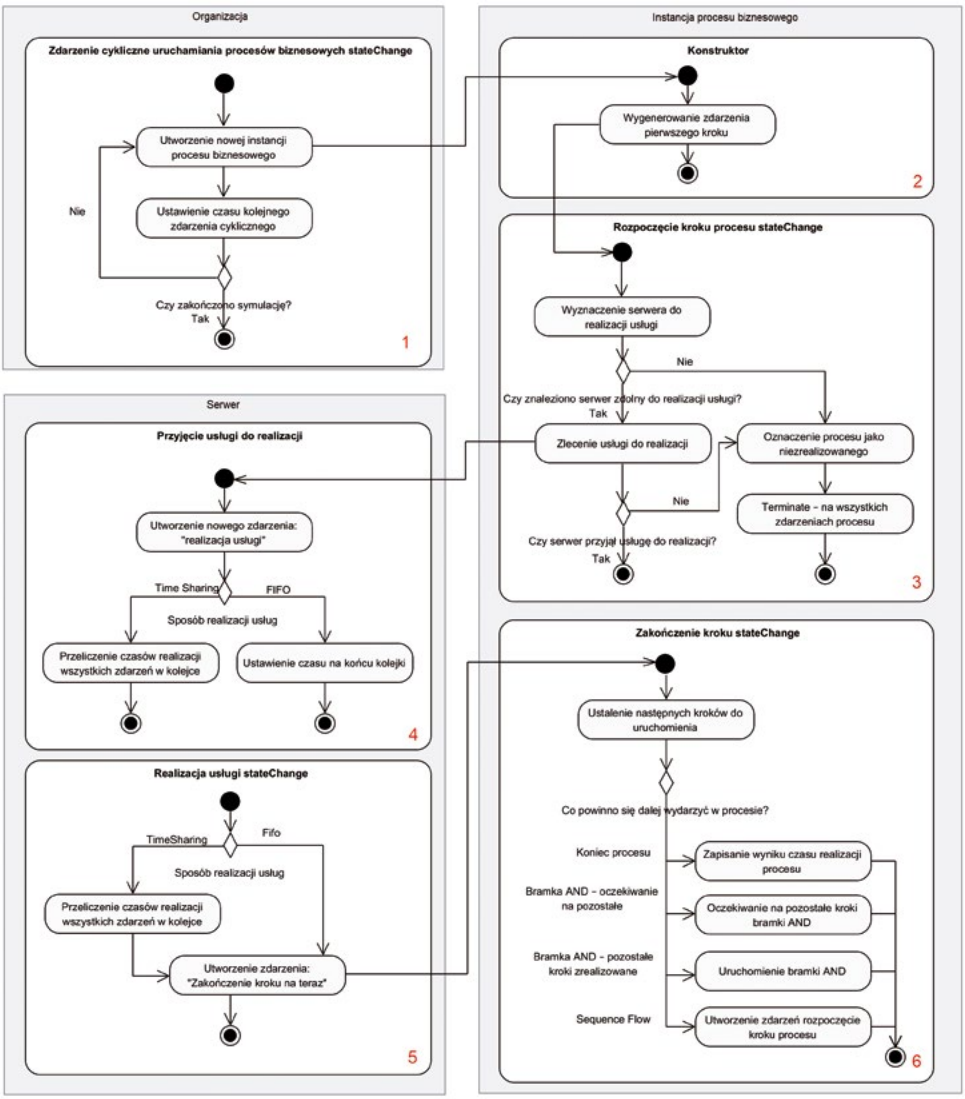
Zdarzenia w symulatorze są względem siebie zależne, a logika ich występowania została przedstawiona na rysunku 2. Po uruchomieniu symulacji generowane jest „Uruchamianie procesów biznesowych". Występuje jedno takie zdarzenie dla każdej definicji procesu biznesowego. Reprezentuje ono utworzenie nowej instancji procesu biznesowego.

Gdy tylko czas symulacji osiągnie czas zdarzenia, to tworzona jest nowa instancja procesu i nowe zdarzenie w kalendarzu, które wystąpi za losowo wygenerowany czas symulacyjny. Czas pomiędzy kolejnymi zdarzeniami uruchomienia procesu jest zdefiniowany za pomocą zmiennej losowej w definicji procesu biznesowego.

4.2. Realizacja procesu biznesowego

Gdy zostanie powołana do życia instancja procesu biznesowego, generowane jest zdarzenie realizacji jego pierwszego kroku z aktualnym czasem symulacyjnym. Każde zdarzenie reprezentujące krok w procesie biznesowym ma na celu powołanie do życia usługi, która ma ten krok zrealizować. W pierwszej kolejności wyznaczany jest właśnie ten serwer, który ma wykonać usługę. Reprezentuje to działanie load balancera, który ma wybrać właściwą instancję komponentu, czyli zrealizować algorytm Service Selection. W symulatorze zostały zaimplementowane dwie strategie Service Selection:

- najpierw serwer najmniej obciążony,
- najpierw serwer o najkrótszym spodziewanym czasie odpowiedzi.



RYS. 2. Algorytm działania symulatora
FIG. 2. The algorithm of the simulator operation

ŹRÓDŁO: opracowanie własne.
SOURCE: own elaboration.

Istnieje oczywiście możliwość rozszerzania symulatora o kolejne algorytmy Service Selection. Jeśli nie zostanie znaleziony serwer zdolny do realizacji kroku w procesie to jest on kończony, a informacja o niezdolności systemu do realizacji procesu odkładana do wyników symulacji.

4.3. Realizacja usługi

Jeśli znaleziono serwer zdolny do realizacji usługi, czyli taki, który:
- ma wystarczającą ilość wolnej pamięci RAM,
- ma uruchomiony komponent zdolny do realizacji usługi,

to tworzone jest zlecenie realizacji usługi. Czas realizacji usługi to suma czasu transferu przez sieć oraz realizacji zadania. Czas transferu jest zależny od wielkości danych do przesłania zdefiniowanych w usłudze oraz przepustowości sieci. Czas realizacji jest zależny od mocy procesora, jaką serwer może przeznaczyć na rzecz realizacji usług, liczby usług zleconych do realizacji, mocy potrzebnej do realizacji usługi oraz modelu działania komponentu (FIFO lub Time Sharing). Jeśli komponent działa w trybie Time Sharing, to każde pojawienie się nowej usługi do realizacji oraz każde zakończenie przetwarzania usługi wymaga przeliczenia oczekiwanych czasów realizacji usług.

4.4. Realizacja kroku procesu

Zakończenie realizacji usługi powoduje utworzenie zdarzenia zakończenia kroku w procesie biznesowym. Ma ono na celu ustalenie kolejnych działań w ramach procesu. Może to skutkować zakończeniem procesu lub utworzeniem zdarzeń rozpoczęcia jednego lub wielu kroków w procesie. Jeśli po kroku w procesie biznesowym użyta została jedna z bramek XOR, OR lub AND, to taki krok ma relacje do wielu kolejnych kroków w procesie. Każda relacja jest opisana prawdopodobieństwem wyboru ścieżki w przypadku bramek XOR i OR. W przypadku bramki XOR wybierany jest dokładnie jeden kolejny krok w procesie, zgodnie z prawdopodobieństwami w relacjach, które sumują się do 1. W przypadku bramki OR prawdopodobieństwo każdej bramki liczone jest niezależnie. Ich suma może być większa niż 1, a więc może zostać wybrany jeden lub wiele kroków. W przypadku bramki AND uruchamiane są zawsze wszystkie następujące kroki. Analogiczne działanie przeprowadza się w przypadku bramek łączących (występujących przed krokiem) – w tym wypadku, w celu uruchomienia kolejnego kroku, wymagane jest najpierw zakończenie się jednego lub wielu kroków poprzedzających.

5. Podsumowanie

Pomimo, że SOA jest obecnie powszechnym sposobem wytwarzania i zarządzania systemami w organizacjach, to wciąż nie została przygotowana metoda optymalizacji przydziału komponentów do serwerów, która uwzględniałaby wszystkie istotne uwarunkowania związane z tym podejściem. Została więc zaproponowana metoda oparata o algorytm genetyczny, który do oceny rozwiązania wykorzystuje symulację. Do realizacji symulatora zastosowano środowisko DISSim ze względu na jego wysoką

wydajność oraz łatwość w implementacji i debugowaniu. Przy projektowaniu symulatora opartego o zdarzenia warto zamodelować działania obiektów symulacyjnych w formie algorytmu, a następnie rozmieścić w nim zdarzenia symulacyjne.

Zaprezentowany symulator pozwala nie tylko na optymalizację przydziału komponentów do serwerów, ale także może być wykorzystywany do analizy projektowanych systemów SOA. Pozwala m.in. na:

- wykrywanie wąskich gardeł,
- ocenę ryzyka niewykonania procesu biznesowego,
- analizę wpływu wprowadzanych w systemie planowanych zmian na procesy biznesowe organizacji.

Może on zostać również wykorzystany do badań naukowych nowych algorytmów wyboru instancji usługi oraz wyboru kolejności realizacji usług w kolejkach komponentu (obecnie zaimplementowano FIFO oraz time sharing).

Literatura

1. Czarnul P. Modelling, optimization and execution of workflow applications with data distribution, service selection and budget constraints in BeesyCluster. *Computer Science and Information Technology* (*IMCSIT*); 2010.
2. Xiang C, Zhao W, Tian C, Nie J, Zhang J. *QoS-aware, Optimal and Automa-ted Service Composition with Users' Constraints*, e-Business Engineering (ICEBE), 2011, IEEE 8th International Conference.
3. Liu Y, Wu L, Liu S. *A Novel QoS-Aware Service Composition Approach Based on Path Decomposition*, Services Computing Conference (APSCC), 2012, IEEE Asia-Pacific.
4. Syu Y, FanJiang Y, Kuo J, Ma S. *Towards a Genetic Algorithm Appro-ach to Automating Workflow Composition for Web Services with Transactional and QoS-Awareness*, IEEE World Congress; 2011.
5. Ludwig S. *Clonal selection based genetic algorithm for workflow service selection,* Evolutionary Computation (CEC), 2012, IEEE Congress.
6. Oh M, Baik J, Kang S, Choi H. An Efficient Approach for QoS-Aware Servi-ce Selection Based on a Tree-Based Algorithm. *Computer and Information Science*. 2008; Seventh IEEE/ACIS International Conference.
7. Zhang C, Chang R, Perng C, So E, Tang C, Tao T. *An Optimal Capacity Planning Algorithm for Provisioning Cluster-Based Failure-Resilient Composite Services*. Services Computing, 2009. SCC IEEE International Conference.
8. Xie L, Luo J, Qiu J, Pershing J, Li Y, Chen Y. *Availability weak point analysis over an SOA deployment framework*. Network Operations and Management Symposium; 2008.

9. Almeida J, Almeida V, Ardagna D, Francalanci C, Trubian M. *Resource Management in the Autonomic Service-Oriented Architecture*, Autonomic Computing, 2006. ICAC IEEE International Conference.
10. Dyachuk D, Deters R. *Service Level Agreement Aware Workflow Scheduling*, Services Computing, 2007. SCC IEEE International Conference.
11. Dyachuk D, Deters R. *Ensuring Service Level Agreements for Service Workflows*, Services Computing, 2008. SCC IEEE International Conference.

Streszczenie

Architektura zorientowana na usługi (SOA) stała się popularna w wielu organizacjach, a pojawienie się koncepcji mikrousług wzmaga proces wdrażania jej w kolejnych firmach. W związku z tym coraz istotniejsze jest wdrażanie systemów SOA w sposób, który jak najlepiej wspiera procesy biznesowe organizacji. W opracowaniu przedstawiono symulator działania systemów SOA, który ma pozwolić na jego ewaluację. Pokrótce opisano zastosowane środowisko implementacyjne – DISSim. Następnie pokazano model klas takiego symulatora oraz główny algorytm, według którego pojawiają się kolejne zdarzenia.

Słowa kluczowe: SOA, symulacja, procesy biznesowe, optymalizacja

Summary
Simulator of SOA class systems

Service-oriented architecture (SOA) has become popular in many organizations, and the emergence of the concept of micro-services strengthens the process of its implementation in companies. Therefore, it is increasingly important to implement SOA systems in a way that best supports the organization's business processes. The paper presents a simulator of the SOA systems, which is to allow its evaluation. The implementation environment is briefly presented. Next, the class model of such a simulator and the main algorithm according to which subsequent events appear is shown.

Keywords: SOA, simulation, business processes, optimization