II Konferencja Naukowa KNWS'05
"Informatyka- sztuka czy rzemiosło"
15-18 czerwca 2005, Złotniki Lubańskie

MODEL STATYCZNY KOMPONENTÓW IMPLEMENTOWANYCH W TECHNOLOGII CORBA COMPONENT MODEL

Tomasz Gratkowski

Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski

t.gratkowski@iie.uz.zgora.pl

STRESZCZENIE

Streszczenie Artykuł prezentuje statyczną reprezentację diagramów komponentów CORBA. Zaproponowane rozszerzenie notacji UML 1.5 umożliwi automatyczne generowanie z diagramów komponentów pliki IDL3, opisujące sposób osadzenia i współdziałania z komponentem CORBA.

1. WPROWADZENIE

Proces budowania systemów informatycznych w oparciu o istniejące, wydajne języki wyższego rzędu, w głównej mierze zorientowane obiektowo, staje się mało wydajny i czasochłonny, szczególnie przy dużych projektach informatycznych oraz przy dużej grupie współpracujących ze sobą informatyków. Programowanie komponentowe, które umożliwia wielokrotne wykorzystywanie dużych wybranych elementów już wcześniej napisanego oprogramowania, redukuje nakład pracy oraz równomiernie rozkłada koszty poniesione na produkcję programu, na większą liczbę realizowanych projektów. Daje również możliwość łatwiejszego przydzielania konkretnym osobom, realizacji poszczególnych fragmentów projektu.

Wiele ośrodków naukowych oraz firm komercyjnych prowadzi zakrojone na szeroką skalę badania nad opracowaniem bardziej uniwersalnej metody implementowania projektów informatycznych z jak najbardziej funkcjonalnych bloków i komponentów [9]. Jednym z nich jest konsorcjum Object Management Group (OMG), które od wielu lat nadzoruje i aprobuje rozwój technologii wspierających rozproszone programowanie zorientowane obiektowo. W roku 1999 rozszerzono model obiektowy Common Object Request Broker Architecture (CORBA) o model komponentów CORBA Component Model (CCM) [6]. Przedstawiona w 2001 roku specyfikacja, opisuje właściwości oraz metody komponentów, deklarowane w języku Interface Definition Language 3 (IDL3) [6], określa sposób osadzenia i komunikacji z komponentem. Wcześniejsze dość ogólne podejście do komponentu zostało bardziej sformalizowane poprzez rozszerzenie właściwości oraz dodanie portów do komunikacji pomiędzy komponentami i portów obsługujących wymianę informacji o zdarzeniach

pomiędzy komponentami. Dla tego rozbudowanego modelu OMG CCM Implementers Group zaprezentował diagramy komponentów COBRA (DKC), które w odróżnieniu od diagramów komponentów języka Unified Modeling Language (UML) [2], bardziej dokładniej przedstawiają wspomniane cechy komponentu CORBA.

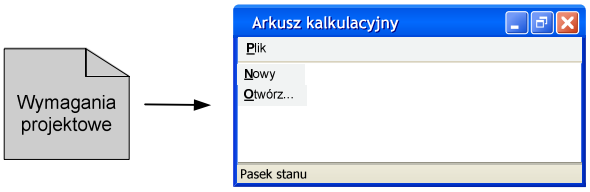
W procesie tworzenia oprogramowania stosowanych jest wiele modeli pomagających podzielić proces produkcyjny na mniejsze, funkcjonalnie spójne fragmenty. Modele te pomagają określić między innymi, kto czym się zajmuje w projekcie, kiedy zaczyna i kończy swoje zadanie oraz co powinno być wynikiem realizacji kolejnego etapu procesu. Analizując rożne modele: model kaskadowy [13], model spiralny [13], model V [7], Rational Unified Process [12], można wyodrębnić, w części modeli, wstępne fazy tworzenia wymagań oraz analizy i projektowania, które poprzedzają fazę kodowania. Artykuł prezentuje początkową fazę prac badawczych służących oszacowaniu wymaganej szczegółowości wymagań projektowych w celu wyeliminowania niejednoznaczności. Umożliwi to efektywniejsze implementowanie aplikacji rozproszonych w środowisku CCM. Końcowym efektem prezentowanego projektu jest opracowanie formalnego zapisu wymagań stawianych aplikacji, na bazie których możliwe będzie automatyczne generowanie interfejs komponentu CORBA w języku IDL 3.

2. INŻYNIERIA OPROGRAMOWANIA

Cel działań grupy informatyków jest bardzo prosty: tworzyć oprogramowanie dobrej jakości, na czas i w ramach wyznaczonego budżetu, zaspakajając rzeczywiste potrzeby klienta [13]. W początkowym etapie rozwoju informatyki, oprogramowywane problemy informatyczne były proste, dlatego używane mechanizmy wspierające informatyków nie były rozbudowane. W miarę jednak wzrostu wymagań i możliwości, jakie oferują systemy informatyczne, wzrosła złożoność technik wspomagających zarządzanie i implementowanie dużych projektów informatycznych. Z tych powodów, realizacja postawionego celu znacznie się wydłużyła i skomplikowała. Istnieje znacznie większy zbiór czynników wpływających na nie zrealizowanie zamierzonego celu. Na przeciw tym negatywnym czynnikom wychodzą nowoczesne techniki inżynierii oprogramowania.

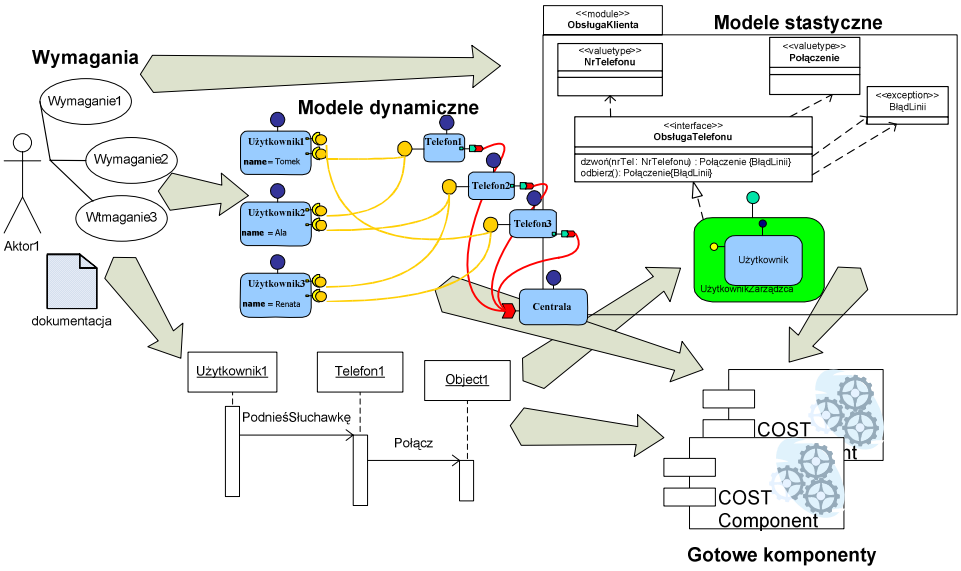
2.1. Wymagania projektowe

W procesie produkcyjnym aplikacji bardzo ważnym etapem jest określenie wymagań projektowych (WP). Poprawne określenie wymagań jest bardzo trudnym i złożonym procesem. Brak wymagań lub ich błędne zrozumienie, jest przyczyną nie zrealizowania zamierzonego celu. Rysunek 2.1 przedstawia idealną sytuację, gdy bezpośrednio z wymagań projektowych następuje bezpośrednie przejście do kodu aplikacji. Dla prostych aplikacji taki scenariusz jest do zaakceptowania, ale dla złożonych rozproszonych systemów takie postępowanie przyczyniłoby się do nie zrealizowania zamierzonego celu.



Rys. 2.1. Przejście z wymagań projektowych od gotowej aplikacji

Rysunek 2.2 przedstawia schemat postępowania przy implementowaniu bardziej złożonych systemów. Na bazie wymagań projektowych przygotowywane są modele dynamiczne oraz statyczne. Odzwierciedlenie wymagań projektowych przy pomocy modeli realizowane jest głównie przez człowieka, przy użyciu dedykowanych narzędzi. Jakość oraz sposób zapisu wymagań w znacznym stopniu wpływa na poprawność przygotowywanych modeli. Dlatego wymagania określone na początku procesu produkcyjnego muszą być kompletne (opisują wszystkie elementy), spójne (wszystkie elementy dopasowane), identyfikowalne (respektują wymagania systemu) oraz testowalne (użyteczne przy testowaniu końcowym) [1]. Złożoność oraz wieloaspektowość wymagań przedstawiono w publikacji [13]. Zamodelowane wymagania projektowe zostają przeniesione na poszczególne komponenty systemu [9, 10, 5]. Kierunek przejścia z wymogów poprzez modele do rzeczywistych komponentów wykorzystuje inżynierię w przód.



Rys. 2.2. Wykorzystanie wymogów projektowych i modeli w procesie implementowanie komponentów

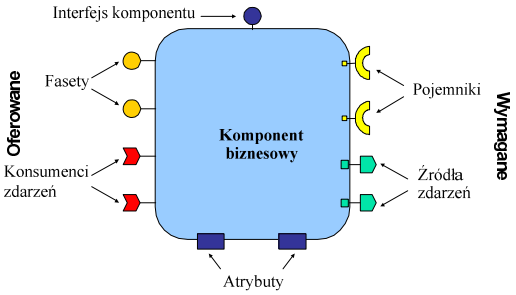
2.2. Modele statyczne i dynamiczne dla komponentów

Najpowszechniej używanym językiem stosowanym w modelowaniu jest język UML [15]. Specyfikacja UML prezentuje bardzo ogólny model statyczny dedykowany dla komponentów

wykorzystujący diagramy komponentów. W publikacjach naukowych można znaleźć wiele przykładów adaptacji innych diagramów (diagramów przepływu [11], diagramów kooperacji, diagramów stanów [16]) w procesie modelowania komponentów. Są to jednak modele bardzo ogólne, nieprecyzujące wszystkich istotnych aspektów, potrzebnych na przykład w procesie projektowania komponentów CORBA.

Dlatego w [6] przedstawiono nowy diagram komponentu CORBA (DKC) rozszerzony o dodatkowe porty (Rysunek 2.3):

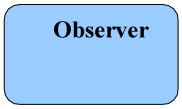
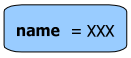
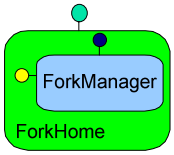
- fasety (ang. facets) – są wyraźnie nazwanymi interfejsami wykorzystywanymi przez klienta do interakcji z komponentem;
- pojemniki (ang. receptacles) – są punktami opisującymi zdolność komponentu do użycia zewnętrznego źródła informacji;
- źródła zdarzeń (ang. event sources) – są punktami wysyłającymi specjalnego typu zdarzenia do jednego lub wielu odbiorców zdarzeń lub kanałów zdarzeń;
- konsument zdarzeń (ang. event sinks) – są punktami, które odbierają zdarzenia specjalnego typu;
- atrybuty (ang. attributes) – nie zabezpieczone nazwane wartości, używane na przykład do konfiguracji komponentu.



Rys. 2.3. Diagram komponentu CORBA (DKC)

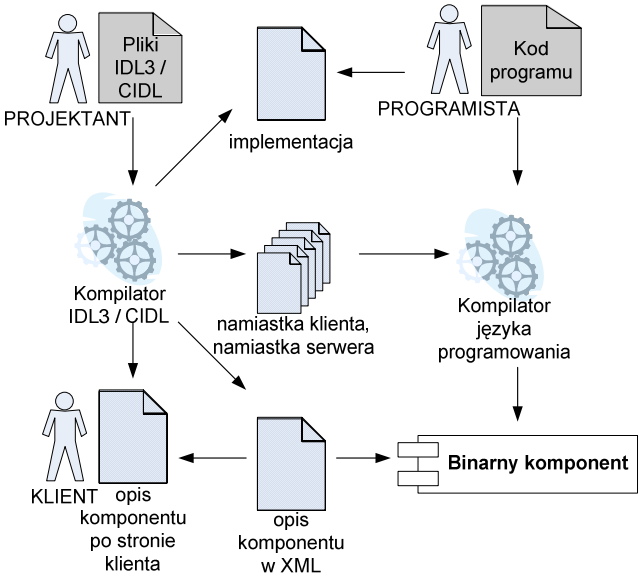
Zaproponowana w CORBA 3.0 formalizacja komponentów wynikała z dużych nieścisłości i dowolności rozwiązań, jakie istniały w wersji CORBA 2.0. Przedstawione porty mają swoje odwzorowanie w języku IDL3. W tabeli 2.1 przedstawiono elementy posiadające odwzorowanie zaproponowane przez OMG. Na bazie przedstawionych informacji możliwe jest modelowanie diagramu przedstawiającego dynamikę projektowanego systemu.

Tab. 2.1. Elementy języka IDL i ich odwzorowanie w DKC, źródło[4]

Nazwa w języku IDL3	Element w DKC	Opis
interface Fork{...};		Interfejs Fork
component ForkManager{...};		Komponent ForkManager
provides Fork the_fork;		Faseta the_fork typu Fork
uses Fork left;		Pojemnik left typu Fork
publishes StatusInfo info;		Źródło zdarzeń info typu StatusInfo
consumes StatusInfo info;		Konsument zdarzeń info typu StatusInfo
attribute string name;		Atrybut name typu string
home ForkHome **manages** ForkManager {};		Określenie domu ForkHome dla komponentu ForkManager

2.3. Projektowanie komponentu CORBA

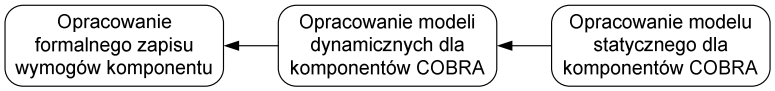
Proces projektowania komponentów CORBA, przedstawiony przez OMG, rozpoczyna projektant (Rysunek 2.4), definiując przy użyciu języka IDL3 interfejsy obiektów wykorzystywanych w procesie implementacji komponentu oraz wskazując relację pomiędzy komponentami używającymi dedykowanych mechanizmów komunikacji międzykomponentowej. Dodatkowo w języku Component Implementation Definition Language (CIDL) może opisać strukturę i stan implementowanego komponentu. Pliki CIDL wykorzystywane są przez kompilator ORB, przy generowaniu kodu namiastek, użytych podczas implementacji komponentu przez programistę. Po kompilacji plików IDL3 oraz CIDL otrzymywany jest kompletny mechanizm komunikacyjny pomiędzy komponentami oraz szkielet wywoływania metod dostępnych w komponentach, zapisany w konkretnym języku programowania. Ostatnią czynnością, wykonaną przez programistę, jest zaimplementowanie ciała metod w wygenerowanych przez kompilator plikach źródłowych.



Rys. 2.4 Cykl życia komponentu CORBA, źródło: [CCMT]

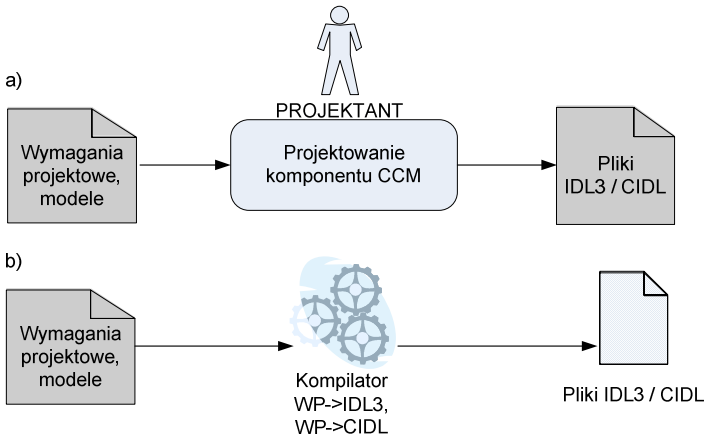
2.4. Od komponentu CORBA do wymogów projektowych

Analizując zaprezentowane fazy wykorzystywane podczas projektowania aplikacji opartej o komponenty CORBA, stosując inżynierię wstecz, zaproponowano przejście od kodu komponentu opisanego przy użyciu języka IDL3, do opisania wymogów komponentu (Rysunek 2.5). Do opisania wymogów zostanie użyta wybrana grupa mechanizmów formalnych w celu wyeliminowania niejednoznaczności specyfikacji z punktu widzenia projektanta systemu i projektanta komponentu CORBA. Zaprezentowane wymagania komponentów powinny realizować wymagania stawiane całemu systemowi. Przeprowadzona analiza ma również wyodrębnić i wskazać, na jakie ważne cechy powinni zwrócić uwagę analitycy tworzący wymagania systemowe, gdy projekt ma zostać zrealizowany w technologii CORBA 3.0.



Rys. 2.5. Harmonogram prac badawczych

Rysunek 2.6a przestawia scenariusz działania, realizowany dzisiaj, oparty o projektanta, który przenosi zamodelowane wymagania projektowe implementując je w języku IDL3.



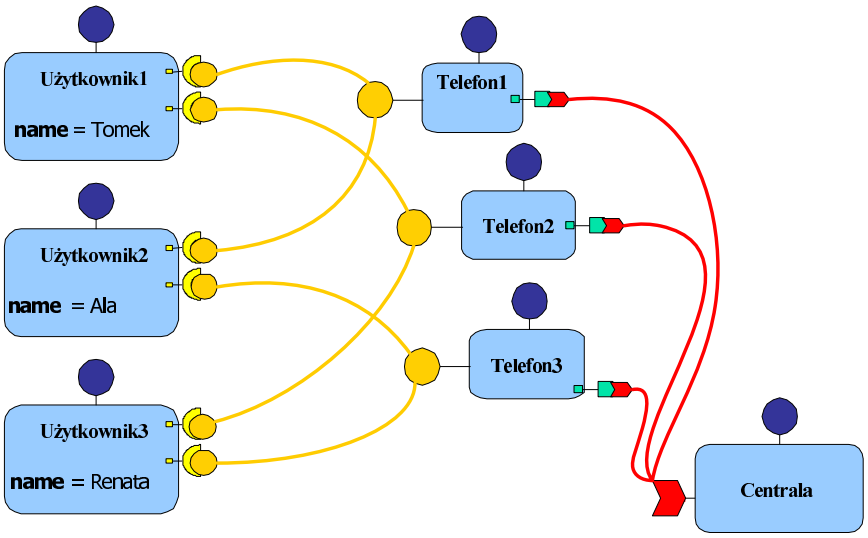
Rys. 2.6. Model wykorzystania WP w procesie projektowania komponentów CORBA

Zformalizowanie zapisu wymogów projektowych oraz opracowanie wymaganych modeli umożliwi realizowanie schematu działań przedstawionego na rysunku 2.6b. Funkcję człowieka przejmie kompilator generujący kod w języku IDL3 z danych wejściowych opisujących wymogi stawiane komponentom CORBA. Oczywiście rola człowieka będzie nie zastąpiona w procesie weryfikacji. Dodatkowo wiedza analityków i projektantów systemu na temat systemów rozproszonych oraz technologii CORBA będzie znacznie większa niż z wykorzystaniem bardziej abstrakcyjnego podejścia oderwanego od architektury projektowanej aplikacji.

3. DIAGRAMY KOMPONENTÓW CORBA

3.1. Dynamiczne diagramy komponentów CORBA

Zaprezentowany przez OMG CCM Implementers Group model komponentu CORBA, dokładniej zaprezentowany w [4] przy użyciu diagramów komponentów CORBA, wykorzystywany jest głównie do przedstawienia dynamicznego współdziałania komponentów. W projekcie Cadena 3], który wykorzystuje środowisko openCCM [14], umożliwiono automatyczne generowanie diagramów dynamicznych CORBA z kodu źródłowego w języku IDL3 w graficznym środowisku programistycznym Eclipse [ECLI2004] (Rysunek 3.1). Taki scenariusz jest możliwy, ponieważ zbiór konstrukcji językowych IDL3 obejmuje zbiór elementów wykorzystywanych na diagramach CORBA oraz konstrukcje IDL3 mają swoje bezpośrednie odpowiedniki na diagramach CORBA (Tabela 2.1).



Rys. 3.1. Dynamiczny diagram komponentów CORBA

Dynamiczny model nie umożliwia jednak automatycznego generowania kodu plików IDL3, ponieważ nie zawiera wszystkich potrzebnych informacji zarówno z punktu widzenia poprawności składni IDL3 jak i dla procesu wprowadzania informacji o komponencie do Repozytorium Interfejsów [6]. Dlatego w procesie projektowania należy wykorzystać diagramy komponentów CORBA wraz z konstrukcjami języka UML, rozszerzone o wszystkie potrzebne informacje, zgodnie z zasadami UML, w celu zautomatyzowania procesu generowania zawartości plików IDL3.

3.2. Statyczne diagramy komponentów CORBA

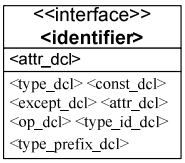
Tabela 3.1 przedstawiono składnię wybranych instrukcji języka IDL3 zapisanych w notacji BNF i proponowane odwzorowanie dla diagramów komponentów CORBA. Zaprezentowano wybraną grupę elementów, użytych w przykładzie prezentującym wykorzystanie statycznego modelu komponentów CORBA. Pełna lista została przedstawiona w [8].

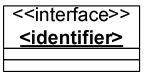
Tab. 3.1. Konstrukcje języka IDL3 oraz odpowiedniki w DKC

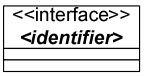
	B N F	D K C
1	<import> ::= “import” <imported_scope> “;” <imported_scope> ::= <scoped_name> | <string_literal>	{“import” <imported_scope>} <imported_scope> ::= <scoped_name> |<string_literal>
2	<type_prefix_dcl> ::= “typeprefix” <scoped_name> <string_literal>	{typeprefix <scoped_name> <string_literal>}
3	<module> ::= “module” <identifier> “{“ <definition>+ “}”	<<module>> <identifier> <definition>+

4

```
<interface> ::= <interface_dcl> | <forward_dcl>
<interface_dcl> ::=  <interface_header> “{”
        <interface_body>“}”
<forward_dcl> ::= [ “abstract” | “local”] “interface”
         <identifier>
<interface_header> ::= [ “abstract” | “local”]
         “interface” <identifier>
[<interface_inheritance_spec>]
<interface_body> ::= <export>*
<export> ::= <type_dcl> “;” | <const_dcl> “;” |
    <except_dcl> “;”| <attr_dcl> “;” | <op_dcl> “;”
        | <type_id_dcl> “;”| <type_prefix_dcl> “;”
```

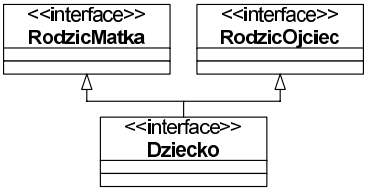






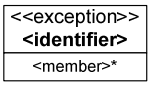
5

```
<interface_inheritance_spec>::=“:” <interface_name>
        { “,” <interface_name> } *
<interface_name> ::= <scoped_name>
<scoped_name> ::= <identifier>| “::” <identifier>
```



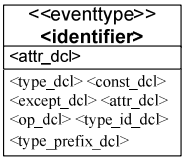
6

```
<except_dcl> ::= “exception” <identifier> “{“
<member>*“}”
 <member> ::= <type_spec> <declarators> “;”
```



7

```
<event> ::= ( <event_dcl> | <event_abs_dcl> |
        <event_forward_dcl>)
<event_dcl> ::= <event_header> “{”
        <value_element> * “}”
<event_header> ::= [ “custom” ] “eventtype”
        <identifier> [ <value_inheritance_spec> ]
<event_abs_dcl> ::= “abstract” “eventtype”
        <identifier> [ <value_inheritance_spec> ]
        “{” <export>* “}”
<event_forward_dcl> ::= [ “abstract” ] “eventtype”
<identifier>
```



8

```
<op_dcl> ::= [ <op_attribute> ] <op_type_spec>
        <identifier> <parameter_dcls>
        [ <raises_expr> ] [ <context_expr> ]
<op_attribute> ::= “oneway”
<op_type_spec> ::= <param_type_spec> | “void”
<parameter_dcls> ::= “(” <param_dcl> { “,”
        <param_dcl> }* “)” | “(” “)”
<param_dcl> ::= <param_attribute>
        <param_type_spec><simple_declarator>
<param_attribute> ::= “in” | “out” | “inout”
<param_type_spec> ::= <base_type_spec>
        | <string_type> | <wide_string_type>
        | <scoped_name>
<raises_expr> ::= “raises” “(” <scoped_name>
```

I D L 3	D K C
in	<simple_declarator> -> <param_type_spec>
out	<simple_declarator> <- <param_type_spec>
inout	<simple_declarator> <-> <param_type_spec>
raises	{<scoped_name>}
context	<<string_literal>>

	{ “,” <scoped_name> } * “)” <context_expr> ::= “context” “(” <string_literal> { “,” <string_literal> } * “)”	<<interface>> **Serwis** wyswietl(tekst : -> string) : void {BladSieci} -> zapisz(tekst : <- int) : void <BladSieci> rejestruj(uzytownik : <-> Uzytkownik) : boolean

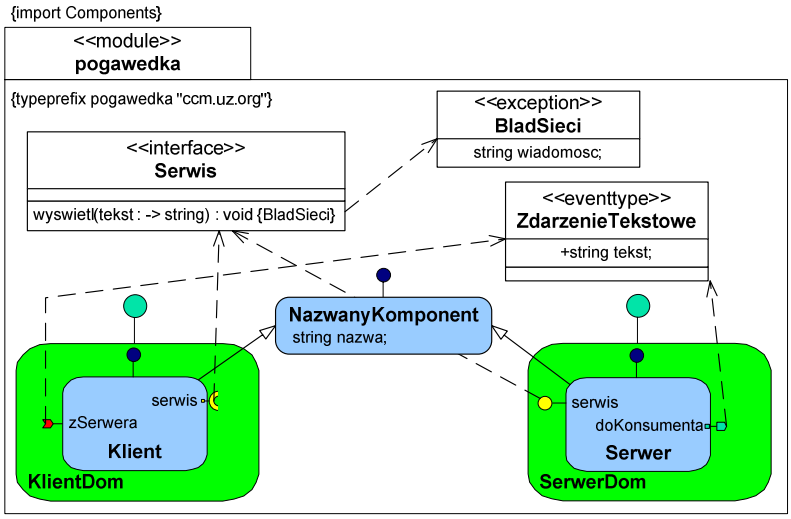
Wykorzystując przedstawione diagramy komponentów CORBA, zamodelowano przykładowe dwa komponenty `Serwer` i `Klient` realizujące serwis pogawędek (Rysunek 3.3). Serwer posiada dwa porty, fasetę `serwis` i źródło zdarzeń `doKonsumenta`, poprzez które może się z nim komunikować klient, posiadający porty pojemnik `serwis` i konsumenta zdarzeń `zSerwera`. Przy użyciu interfejsu `Serwis` wymieniane są dane wykorzystując mechanizm wywołania procedury na komponencie. Natomiast porty `doKonsumenta` i `zSerwera` wykorzystują mechanizm zdarzeń w celu przesłania informacji, wysyłając zdarzenie `ZdarzenieTekstowe`.

Na bazie zamodelowanego statycznego diagramu komponentów CORBA, możliwe jest wygenerowanie przedstawionego na rysunku 3.2 kodu źródłowego w języku IDL3. Automatycznie kompilator staje się narzędziem weryfikującym poprawność specyfikacji przedstawionego modelu.

```
import Components;
module pogawedka{
   typeprefix pogawedka "ccm.uz.org";
   interface Serwis{
     void wyswietl(in string tekst)raises(BladLinii);
   };
   eventtype ZdarzenieTekstowe{
     public string tekst;
   };
   component NazwanyKomponent{
     attribute string nazwa;
   };
   exception BladSieci{
     string wiadomosc;
   };
   component Serwer : NazwanyKomponent{
     provides Serwis serwis;
     publishes ZdarzenieTekstowe doKonsumenta;
   };
   home SerwerDom manages Serwer{
```

```
        };
        component Klient : NazwanyKomponent{
          uses Serwer serwis;
          consumes ZdarzenieTekstowe zSerwera;
        };
        home KlientDom manages Klient{
        };
    };
```

Rys. 3.2. Kod w języku IDL3 komponentu CORBA



Rys. 3.3. Model statyczny dla komponentów CORBA

4. WNIOSKI I KIERUNKI DALSZYCH PRAC

Przedstawiony statyczny model diagramów komponentów CORBA, rozszerzający standard UML 1.5 zgodnie z przyjętymi w nim zasadami, umożliwi automatyczne generowanie definicji komponentu CORBA zapisanej w języku IDL3. Proponowane rozwiązanie wymusi na projektancie systemu większą znajomość zagadnień związanych z programowaniem rozproszonych komponentów, ale jednocześnie zwiększy funkcjonalność przygotowywanych modeli.

Prezentowany artykuł przedstawia początkowy etap badań, zmierzających do określenia wymaganej szczegółowości i sposobu zapisu wymogów projektowych. Przyjęta metodologia postępowania wykorzystuje mechanizm inżynierii wstecz. Dlatego opracowanie dodatkowych modeli, umożliwi w dalszych badaniach przejście do analizy wymagań projektowych.

BIBLIOGRAFIA

[1] B. Boehm: *Requirements that handle IKIWISI*, COTS, and Rapid Change, IEEE Computer, Vol. 33, Issue 7, July 2000, ss. 99-102

[2] G. Booch, J. Rumbaugh, I. Jacobson: *UML przewodnik użytkownika*, Wydawnictwa Naukowo-Techniczne, 2001

[3] projekt *Cadena*, http://cadena.projects.cis.ksu.edu/

[4] P. Merle P, et al.: *CORBA Component Model Tutorial*, OMG Meeting, Orlando, USA, June 25th, 2002

[5] L. Chung, K. Cooper: *Towards a Model Based COTS-Aware Requirements Engineering Process*, 1st International Workshop on Model-Based Requirements Engineering, San Diego, November 2001

[6] *CORBA Components Version 3.0*, Object Management Group, June 2002, http://www.omg.org/technology/documents/formal/components.htm

[7] pod redakcją J. Górski: *Inżynieria oprogramowania w projekcie informatycznym*, MIKOM, 2000

[8] T. Gratkowski: *Model statyczny komponentów implementowanych w technologii CCM*, http://www.iie.uz.zgora.pl/~tgratkow/phd/mskc.html

[9] G. T. Heineman, W. T. Councill: *Component-base software engineering: Putting the Pieces Together*, Addison-Wesley, 2001

[10] C. Ncube, N. A. M. Maiden: *PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm*, 2nd International Workshop on Component-Based Software Engineering, Los Angels, May 1999

[11] P. Kruchten: *Modeling Component Systems with the Unified Modeling Language*, Proceedings of the 1998 International Workshop on Component-Based Software Engineering, http://www.sei.cmu.edu/cbs/icse98/papers/p1.html

[12] P. Kruchten: *The Rational Unified Process An Introduction*, Second Edition, Addison-Wesley, 2000

[13] D. Leffingwell, D. Widrig: *Zarządzanie wymaganiami*, Wydawnictwa Naukowo-Techniczne, 2003

[14] projekt *OpenCCM - The Open CORBA Component Model Platform*, http://openccm.objectweb.org/

[15] *OMG Unified Modeling Language Specification*, Version 1.5, formal/03-03-01, Object Management Group, 2003

[16] A. Wienberg, F. Matthes, M. Boger: *Modeling Dynamic Software Components in UML*, UML'99 - The Unified Modeling Language. Proceedings of the Second International Conference, Fort Collins, USA, 1999