Zastosowanie Promeli do weryfikacji aplikacji opartych na środowisku CORBA

Jarek Bąbel

Wstęp

1. Technologia obiektów rozproszonych
   - oprogramowanie pośrednie (ang. middleware)
   - różnorodność środowiska rozproszonego
   - CORBA
2. Problemy
   - pomijanie różnicy między lokalnymi i zdalnymi obiektami (przezroczystość)
   - przykład: blokowanie zdalnego obiektu
3. Formalna weryfikacja
   - trudność w opanowaniu języka formalnej specyfikacji
   - graficzna reprezentacja systemu
   - automatyczna translacja specyfikacji systemu

Adaptacja podzbioru języka UML

1. System CUP
2. Zwięzłość modelu
3. Elementy CORBY potrzebne do weryfikacji:
   - połączenie klienta ze zdalnym obiektem (ang. binding)
   - strategia przydziału wątków do obsługi komunikacji ze zdalnymi obiektami (ang. thread policy)
   - tryb synchronizacji wywołania zdalnych metod (ang. synchronization mode)
4. Abstrakcja CORBY
5. Zmniejszenie przestrzeni stanów

Wprowadzenie do systemu CORBA

1. CORBA = Common Object Request Broker Architecture
2. Interfejsy IDL
3. ORB
   - połączenie klienta ze zdalnym obiektem (ang. binding)
4. POA
   - wątek podległy (ang. servant)
   - strategia wielowątkowości (ang. thread policy)
5. typy zdalnych wywołań metod
   - synchroniczne
   - asynchroniczne
   - synchroniczne z opóźnionym odbiorem

Przykład

1. System aukcyjny, zasady
2. Wykorzystanie technologii CORBA
3. Własności, które możemy sprawdzić
   - brak blokad
   - licytacja kiedyś się skończy
   - tylko jeden uczestnik wygra
   - wygrywający ma ofertę większą lub równą pozostałym
   - wygrywająca oferta nie może być większa niż ustalony limit
4. Konsekwencje wykorzystania oprogramowania pośredniego

Adaptacja diagramów UML

1. Założenia
   - części składowe modelu:
     - zbiór aktywnych obiektów
     - obiekty ORB i POA
     - obiekty zdalne
   - nie rozważamy dynamicznego tworzenia i niszczenia obiektów
   - każdy obiekt i komponent ma przypisany unikalny, publicznie znany identyfikator
2. Wykorzystane diagramy:
   - wdrożenia
   - klas
   - stanów

Diagram wdrożenia

1. obiekty i komponenty z unikalnymi identyfikatorami
2. relacja wdrożenia między zdalnymi obiektami i POA, między POA i ORB
3. specyfikacja strategii wielowątkowości dla POA i ORB przez umieszczenie obiektu Policy w POA i ORB

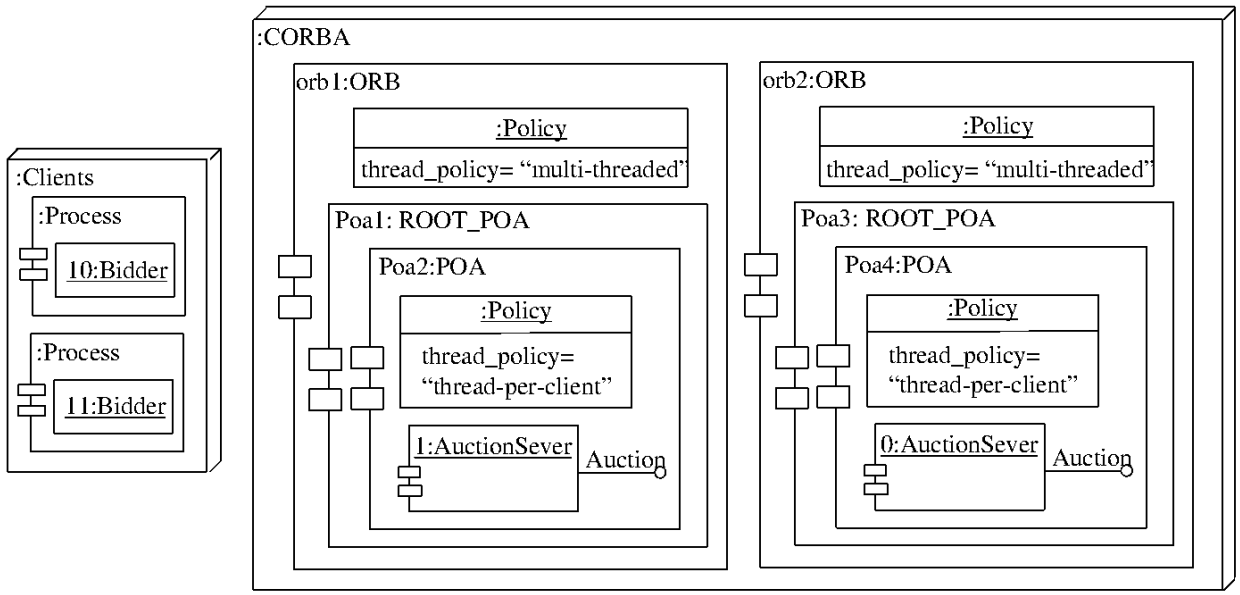
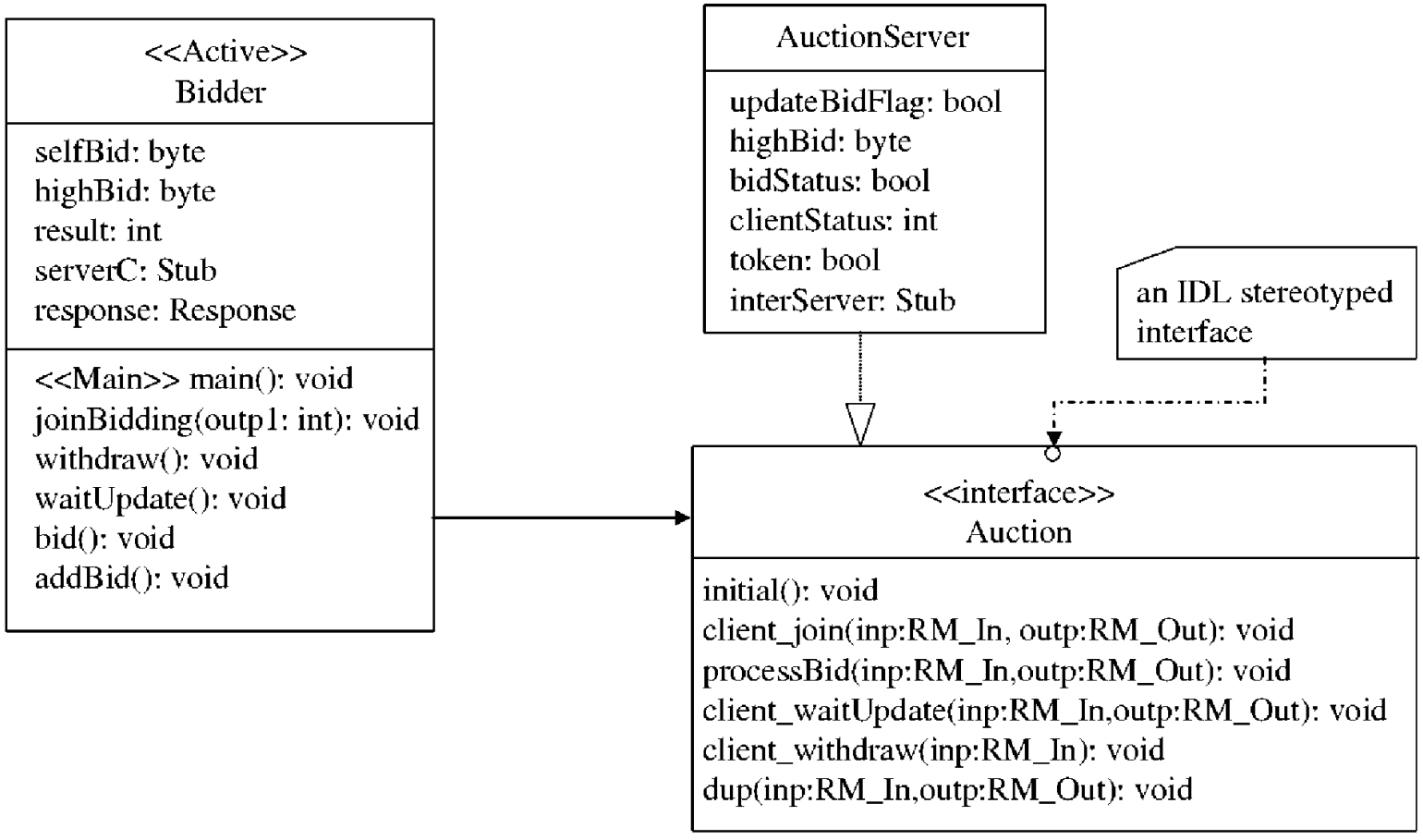


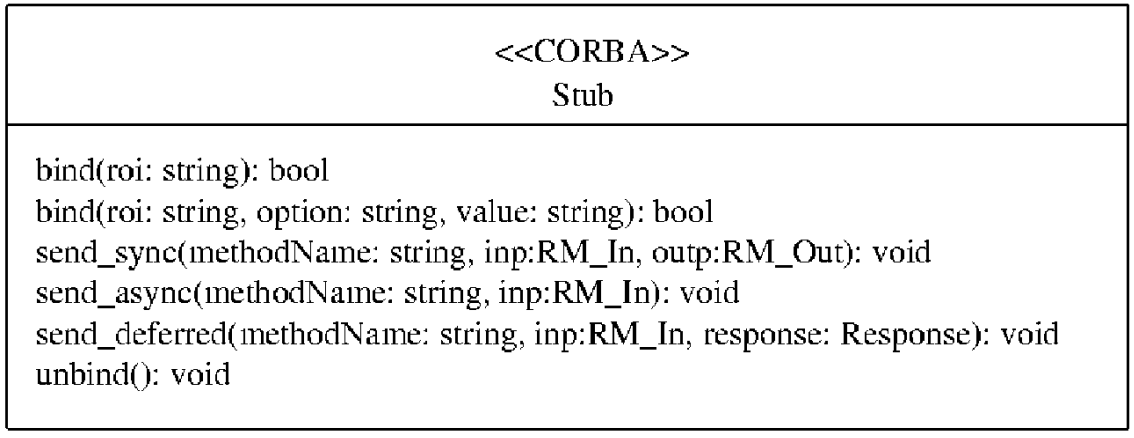
Diagram klas

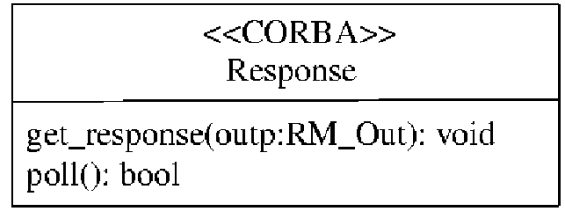
1. Specyfikuje
   - obiekty aktywne
   - interfejsy obiektów zdalnych i ich implementacje
2. Stereotypy metod
   - metoda główna
   - metoda wątku
3. Unifikacja parametrów metod zdalnych
   - brak wartości zwracanej
   - parametry wejściowe i wyjściowe muszą być obiektami klas danych RM_In i RM_Out

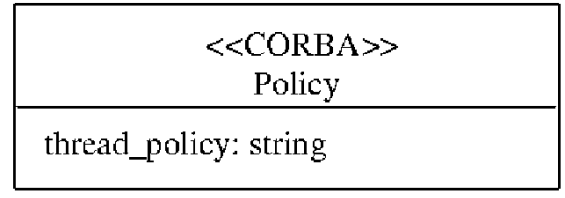
Przykład diagramu klas



Klasy predefiniowane



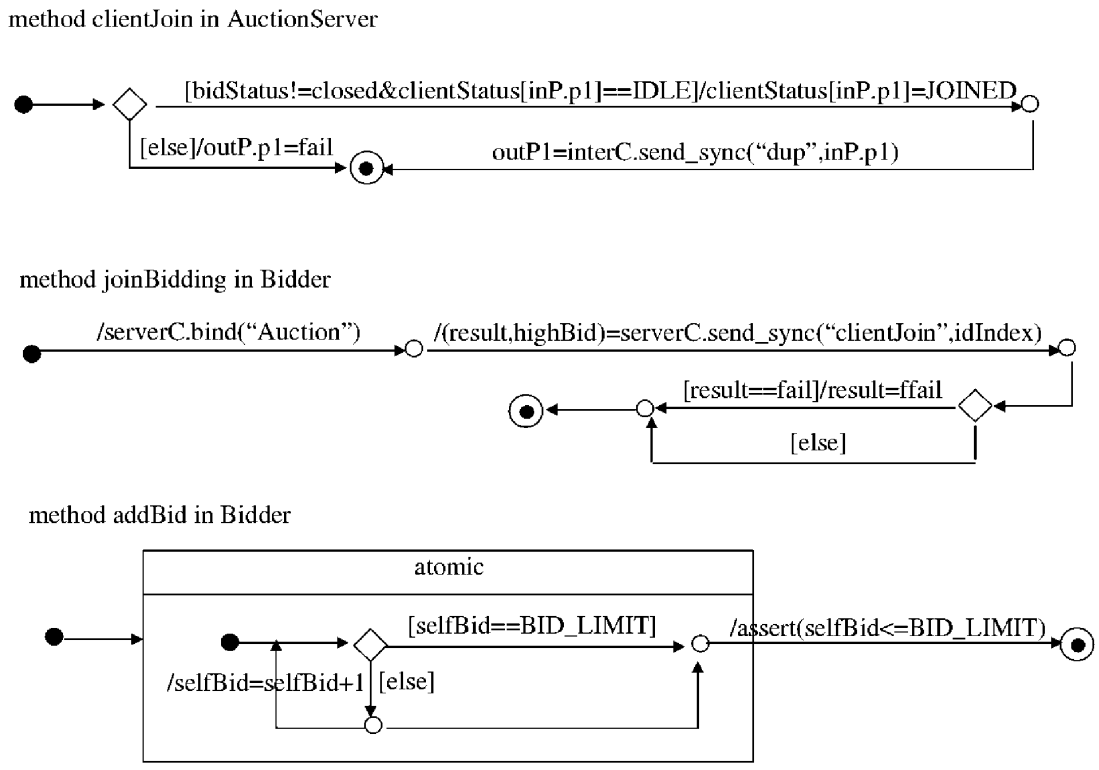




1. Stub
2. Response
3. Policy

Diagram stanów

1. Eliminacja elementów współbieżnych
2. Bloki atomowe
3. Etykiety end, progress, accept



Model POA

1. POA
   - globalny kanał przechowujący żądania zdalnych wywołań metod
   - zbiór wątków podległych (odpowiedni do strategii wielowątkowości)
   - wątek podległy
     - pobiera żądanie
     - wywołuje odpowiednią metodę inline
     - zwraca wynik przez kanał podany w żądaniu

Model ORB

1. ORB
   - jednowątkowy
     - globalny kanał przechowujący żądania
     - proces rozdzielający: pobiera kolejne żądanie i przesyła je do kanału odpowiedniego POA
   - wielowątkowy
     - brak osobnej implementacji
     - żądania są przekazywane bezpośrednio do kanałów POA

Aktywne i zdalne obiekty

1. Aktywny obiekt
   - dane (zdefiniowany typ)
   - zbiór metod typu inline (jedna dla każdej metody obiektu)
   - zbiór procesów Promeli (jeden dla każdej metody typu Main lub Thread)

   Metoda typu Main jest uruchamiana przez proces init Promeli.
2. Zdalny obiekt
   - dane (zdefiniowany typ)
   - zbiór metod typu inline

   Co najmniej jeden wątek POA będzie obsługiwał zdalne wywołania odnoszące się do tego obiektu.

Model Stub i Response

Typy danych w Promeli opisujące klasy *Stub* i *Response*:

```
typedef CUP_Stub
{
    int stubID, objID;
    chan invocation;     /* for sending remote method calls */
    /* return channel : outputParameters */
    chan response = [0] of {CUP_RM_Out};
}
```

```
typedef CUP_Response    /* for deferred method calls */
{
    chan responseChan = [1] of {CUP_RM_Out};    /* outputParameters */
}
```

Związanie obiektu *Stub*

- jeden osobny proces *cup_bind* Promeli (nie związany z żadnym ORB)
- każde żądanie jest obsługiwane przez atomowy blok
- żądania i ich wyniki przesyłane są przez globalne kanały:

```
chan cup_bind_request = [0] of {mtype, mtype, mtype, int};
chan cup_bind_result = [0] of {chan, int};
```

- Proces *cup_bind* interpretuje żądania oraz:
  - odnajduje zdalny obiekt odpowiedni do opcji wiązania
  - wstawia do *cup_bind_result* identyfikator zdalnego obiektu i kanał wywoływania (do dalszej komunikacji z obiektem)
  - opcjonalnie uruchamia wątek podległy

Zdalne wywołanie metody

1. wysłanie żądania do odpowiedniego kanału wywoływania
2. żądania są pobierane z kanału wywoływania i obsługiwane przez proces podległy POA
3. funkcjonalność i ilość wątków podległych POA zależy od strategii wielowątkowości POA
4. obsługa polega na wywołaniu metody inline z odpowiednimi parametrami i przekazanie wyniku do kanału zwrotnego podanego w żądaniu

Przykładowa weryfikacja

1. procesor Pentium(R) 4 3.06 GHz, pamięć 512 Mb, system Windows XP
2. testy na blokadę (2 licytujących):
   - exhaustive 160 min
   - supertrace 7 min (coverage 99%)

   nie wykryto blokady
3. liczba stanów systemu gwałtownie wzrasta po zwiększeniu ilości licytujących (limit licytacji 3):
   - uczestnicy 2, stany 5e+006
   - uczestnicy 3, stany 3.5e+008

Przykładowa formuła LTL

- tylko jeden z licytujących wygrywa:

```
p = ((Bidder[0].result == fsucc) &&
     (Bidder[1].result == ffail))
q = ((Bidder[1].result == fsucc) &&
     (Bidder[0].result == ffail))
LTL: <>(p^q)
```

- błąd: obaj licytujący przegrywają
- przyczyna: uczestnik może zrezygnować z licytacji nie oddając żadnego głosu