Opracowanie do slajdów

Jarek Bąbel

22 czerwca 2005

Spis treści

1 Wstęp

Slajdy i towarzyszące im opracowanie powstały na podstawie pracy autorów J. Chen i H. Cui pt. "Translation from Adapted UML to Promela for Corba-Based Applications". Pracę można pobrać pod adresem:
`html://citeseer.ist.psu.edu/chen04translation.html`

1.1 Technologia obiektów rozproszonych

Współczesne systemy rozproszone często korzystają z oprogramowania pośredniego (ang. middleware), które pozwala na „ukrycie” różnorodności sprzętu i oprogramowania środowiska rozproszonego oraz daje zunifikowany interfejs komunikacji dla rozproszonych aplikacji. Dzięki temu projektowanie i realizacja systemu są znacznie łatwiejsze.

Rozproszone aplikacje napisane w technologii obiektowej zazwyczaj korzystają z oprogramowania pośredniego typu Distributed Object System, dzięki któremu obiekty są dostępne poza granicami procesów uruchomionych na pojedynczym komputerze.

Szeroko akceptowanym standardem dla systemów DOS jest CORBA (z firmy OMG).

1.2 Formalna weryfikacja

Najczęściej programy są modelowane przy użyciu graficznej reprezentacji np. języka UML dla technologii obiektowej.

Typowy programista nie korzysta z formalnych metod weryfikacji ze względu na złożoność zagadnienia formalnego modelowania systemu.

Jednym ze sposobów na ułatwienie wykorzystania narzędzi weryfikacji przez twórców oprogramowania jest automatyczne tłumaczenie graficznej reprezentacji systemu na język formalnej specyfikacji.

W niniejszej pracy opisujemy narzędzie CUP generujące kod Promeli z dostosowanych diagramów UML zapisanych w formacie XML.

2 Adaptacja podzbioru języka UML

Formalna weryfikacja jest ograniczona przez możliwości (głównie ilość pamięci) komputera. Stąd model weryfikowanego systemu musi być zwięzły jednocześnie nie upraszczając istotnych elementów.

Prezentowany system korzysta z następującego podzbioru języka UML:

- diagramy klas

- diagramy stanów
- diagramy wdrożenia

Modelujemy tylko te aspekty CORBY, które mają wpływ na poprawność aplikacji w sensie systemu rozproszonego. Jako interfejs CORBY definiujemy w języku UML następujące elementy:

- odpowiednie wzorce (ang. stereotypes) klas, metod i interfejsów
- predefiniowane klasy dla:
  - zdalnych wywołań metod
  - strategii wykorzystania wielowątkowości
  - połączenia klienta ze zdalnym obiektem
- predefiniowane typy komponentów CORBY:
  - POA
  - ORB

Dzięki tym definicjom uzyskujemy jasny model systemu, w którym CORBA jest dostępna przez jednolity interfejs i jest automatycznie dołączana do modelu w Promeli.

2.1 Ograniczenie przestrzeni stanów

Większość metod zostaje przekształcona na funkcje typu inline w Promeli. Rezygnujemy z elementów współbieżnych w diagramie stanów (sync, fork, join i obszary współbieżne w stanach złożonych). Zamiast nich używamy wzorca określającego metody działające w osobnych wątkach (jeden dla każdej metody).

Dodatkowo w diagramie stanów używane są nazwane stany złożone obejmujące tranzycje wpływające tylko na zmienne lokalne. W ten sposób możemy wykorzystać pojęcie atomowości bez wpływu na prawdziwość wyników weryfikacji i jednocześnie istotnie zmniejszyć przestrzeń stanów.

3 Wprowadzenie do technologii CORBA

3.1 Interfejs IDL

W sposób niezależny od platformy i języka programowania opisuje sposób wykorzystania zdalnego obiektu.

3.2 ORB

Object Request Broker. Podstawowy element CORBY (może być ich wiele, mogą komunikować się ze sobą). Wyszukuje zdalny obiekt do obsługi żądania klienta, zgłasza zdalne wywołanie metody do zdalnego obiektu, przekazuje klientowi wynik zdalnej metody. ORB może być jedno lub wielowątkowy.

Możliwe sposoby łączenia klienta ze zdalnym obiektem:

- żądanie połączenia z konkretnym zdalnym obiektem (potrzebny jest unikalny identyfikator obiektu)
- żądanie połączenia ze zdalnym obiektem podanego typu (potrzebny jest unikalny identyfikator typu)
- żądanie połączenia z obiektem podanego typu zarządzanym przez określony POA (dodatkowo potrzebny identyfikator POA)

3.3 POA

Portable Object Adapter. POA jest interfejsem pomiędzy zdalnym obiektem i ORB. Aktywuje i usypia podległe wątki obsługujące żądania skierowane do obiektu. W ten sposób zarządza zasobami po stronie serwera.

POA działa według ustalonego zbioru zasad. Nas interesuje realizacja wielowątkowości POA, która ustala model wykorzystywany do obsługi wątków podległych. Możliwe realizacje:

- *główny wątek* wszystkie POA związane z ORB dzielą jeden wątek udostępniony przez ORB do obsługi wszystkich wywołań metod
- *osobny wątek dla każdego POA* POA tworzy jeden wątek podległy do obsługi wszystkich wywołań metod
- *zbiór n wątków* POA tworzy zbiór n wątków podległych do obsługi wywołań metod
- *osobny wątek dla każdego obiektu* POA tworzy podległy wątek dla każdego obiektu, którym zarządza
- *osobny wątek dla każdego klienta* POA tworzy podległy wątek do obsługi wywołań metod zgłaszanych przez klienta do danego obiektu
- *osobny wątek dla każdego żądania* POA tworzy wątek podległy dla każdego zdalnego wywołania metody

Zdalne wywołania metod mogą korzystać z tego samego wątku ORB lub POA (w zależności od modelu wielowątkowości) stąd w systemie korzystającym z CORBY mogą powstać blokady.

4 System aukcyjny (przykład)

4.1 Zasady

W systemie aukcyjnym wielu klientów współzawodniczy o wygraną. Uczestnik aukcji może przestać licytować w dowolnym momencie. Licytacja kończy się jeżeli wszyscy licytujący przestaną licytować lub kiedy zostanie osiągnięty ustalony limit. Wygrywa osoba, która po zakończeniu miała najwyższą ofertę.

Przyjmujemy, że system obsługujący licytację składa się z dwóch serwerów pracujących równolegle. Każdy z nich przechowuje lokalną kopię stanu licytacji. Uczestnik aukcji może łączyć się z każdym z serwerów, ale tylko z jednym na raz.

4.2 Zastosowanie CORBY

Serwerom odpowiadają dwa zdalne obiekty. Dla utrzymania spójności wiedzy o licytacji serwery synchronizują się przez zdalne wywołania metod. Pierwszeństwo w aktualizacji stanu licytacji ma serwer, który aktualnie posiada znacznik (ang. token).

Uczestnik licytacji jest reprezentowany przez aktywny (automatycznie uruchamiany) obiekt. Obiekt uczestnika łączy się z systemem aukcji przez obiekt klienta, który przyłącza się do losowo wybranego serwera. Klient licytuje przez wywołanie zdalnej metody na serwerze aukcyjnym, do którego jest podłączony.

4.3 Własności, problemy

Przedstawiony system działa poprawnie dla wielowątkowego ORB i w modelu wątków POA „osobny wątek dla każdego klienta", ale po zastosowaniu modelu wątków POA „osobny wątek dla każdego obiektu" występuje blokada. Przykład: licytujący b1 wysyła żądanie do serwera s1, w tym samym czasie b2 wysyła żądanie do s2. Teraz s1 próbuje wywołać zdalną metodę w s2, żeby zsynchronizować stan aukcji. Podobnie s2. Obydwie próby wywołania zdalnych metod przez s1 i s2 są blokowane, ponieważ jedyny wątek obsługujący s1 jest zablokowany przez b1 i podobnie jedyny wątek s2 jest zablokowany przez b2. b1 i b2 nie zwolnią wątków s1 i s2 dopóki nie otrzymają potwierdzenia swoich żądań. W ten sposób cały system zostaje zablokowany. Warto zauważyć, że takich problemów nie da się wykryć przy modelowaniu zdalnych wywołań metod w taki sam sposób jak lokalnych.

5 Adaptacja diagramów UML

5.1 Wykorzystane diagramy

Diagram wdrożenia opisuje tworzenie i wdrożenie (wzajemne relacje) wszystkich instancji obiektów aplikacji. Diagramy klas i stanów opisują specyfikacje obiektów. Każda metoda zdefiniowana na diagramie klasy jest związana z diagramem stanów, który opisuje jej działanie. Zachowanie klasy jest wyspecyfikowane przez zbiór diagramów stanów jej metod.

Diagramy stanów składają się z:

- sparametryzowanych wywołań metod
- zdarzeń zmiany (np. wciśnięcie klawisza)
- warunków (ang. guard conditions)
- przypisań wartości

5.2 Diagram wdrożenia

Składa się z dwóch elementów:

- Element związany z CORBĄ (strona serwera aplikacji) zawiera co najmniej jeden komponent ORB. Każdy element ORB zawiera główny komponent POA i opcjonalnie obiekt Policy definiujący model wielowątkowości dla ORB (jeżeli nie ma tego obiektu, to przyjmuje się strategię jednowątkową).

  Każdy element POA zawiera zero lub więcej komponentów zdalnych obiektów i komponentów POA. Zdalne obiekty są przedstawione jako komponenty implementujące interfejs zdalnego obiektu (IDL). POA może zawierać obiekt definiujący strategię wielowątkowości. Brak takiego obiektu oznacza dziedziczenie strategii z POA nadrzędnego. Główny element POA zawsze ma model wielowątkowości typu „główny wątek".

- Element związany ze stroną klientów systemu zawiera zbiór komponentów Process. Każdy element typu Process zawiera aktywny obiekt, który jest automatycznie uruchamiany podczas startu systemu.

5.3 Diagram klas

Dla wyróżnienia roli jaką pełni klasa w aplikacji definiujemy wzorzec klasy Active dla klas aktywnych obiektów i wzorzec interfejsu IDL dla interfejsów zdalnych obiektów.

Obiekt aktywny posiada metodę główną spełniającą wzorzec Main, która jest automatycznie uruchamiana. Dodatkowo definiujemy wzorzec metody typu Thread. Metoda o wzorcu Thread po wywołaniu zostaje uruchomiona w osobnym wątku. Taka metoda nie może zwracać wartości.

5.4 Stub, Response i Policy

Stub i Response, to predefiniowane klasy związane z systemem CORBA. Ich zadaniem jest obsługa zdalnych wywołań metod. Obiekt typu Stub reprezentuje klienta zdalnego obiektu (np. pole serverC w klasie aktywnego obiektu Bidder jest ustawiane w odpowiednim diagramie stanów na zdalny obiekt typu Auction).

Obiekt klasy Response służy do opóźnionego odczytywania wyniku wywołania zdalnej metody (ang. deferred remote method call).

5.4.1 Stub

Obiekt Stub staje się klientem zdalnego obiektu przez wywołanie odpowiedniej metody połączeniowej (ang. binding). Po zestawieniu połączenia według ustalonych parametrów zdalne wywołania metod obiektu można realizować przez metody obiektu Stub:

- send_sync
- send_async
- send_deferred

Wywołania zdalnej metody różnią się sposobem odbioru wyniku. Odpowiednio: oczekiwanie na wynik, brak wyniku, odebranie wyniku po jakimś czasie przy pomocy obiektu typu Response.

Obiekt Stub może być odłączony od zdalnego obiektu (przez metodę unbind) i ponownie wykorzystany do komunikacji z innym obiektem.

5.4.2 Policy

Zawiera napis określający strategię wielowątkowości POA lub ORB. Zakładamy, że wielowątkowy ORB ma nieograniczoną ilość zasobów i nie blokuje się. Nie rozważamy komunikacji między różnymi ORB.

5.5 Diagram stanów

(Patrz 2.1) Stany, których nazwy zaczynają się od słów: *end*, *progress* lub *accept* są przekształcane na odpowiedniki w Promeli.

Do tranzycji można dodawać asercje. Przykładowo na diagramie metody addBid asercja pozwala sprawdzić, czy przyjęta oferta uczestnika licytacji nie przekracza ustalonego limitu.

6 Model w Promeli

Klasy danych RM_In i RM_Out są przekształcane do dwóch typów *CUP_RM_In* i *CUP_RM_Out*.

6.1 Stub

Klasa Stub jest reprezentowana przez typ CUP_Stub, który zawiera dwa kanały:

- *invocation* do wysyłania zdalnych wywołań metod (kanał wywołania)
- *response* do opóźnionego odbierania wyniku wywołania metody

i dwie wartości całkowite *objID* oraz *stubID* wykorzystywane do ustalenia połączenia.

6.2 Response

Klasa Response reprezentowana jest przez typ CUP_Response zawierający kanał służący do opóźnionego odbierania wyników zdalnego wywołania.

6.3 Związanie obiektu Stub

Żądanie zestawienia połączenia ze zdalnym obiektem umieszczane jest w kolejce *cup_bind_request* i zawiera następujące elementy:

- unikalny identyfikator obiektu Stub
- typ zdalnego obiektu
- typ wiązania (CUP_RANDOM, CUP_POA, CUP_OBJ)
- identyfikator POA lub zdalnego obiektu

Znaczenie typów wiązania;

- *CUP_RANDOM* związanie z losowym obiektem implementującym podany interfejs (typ zdalnego obiektu)

- CUP_POA związanie z obiektem implementującym podany interfejs w danym POA
- CUP_OBJ związanie z konkretnym obiektem implementującym podany interfejs

6.3.1 Translacja wiązania obiektu Stub

```
stub.bind("Auction")
```

Zostaje przekształcone na:

```
cup_bind_request!stub_T.stubID, CUP_IDL_Auction,CUP_RANDOM, scratch;
cup_bind_request?stub_T.invocation, stub_T.obj_ID;
```

W trakcie translacji mogą ulegać zmianie nazwy zmiennych (w tym wypadku dla jednoznaczności dodane T (jak Translated) do nazwy stub).

6.4 Zdalne wywołanie metody

Procesy podległe POA pobierają żądania z kanału wywołania i obsługują je. Liczba i zachowanie procesów podległych zależy od strategii wielowątkowości POA (patrz 3.3):

- jeden proces podległy jest tworzony dla każdego ORB, który posiada co najmniej jedno POA z modelem wielowątkowości typu *główny wątek*
- wiele procesów podległych jest tworzonych dla każdego POA, które ma strategię inną niż *główny wątek*; te procesy obsługują żądania skierowane do kanału konkretnego POA; możliwe realizacje:
  - *osobny wątek dla każdego POA* jeden proces do obsługi wszystkich żądań kierowanych do POA
  - *zbiór n wątków* n procesów do obsługi wszystkich żądań kierowanych do POA
  - *osobny wątek dla każdego obiektu* jeden proces do obsługi wszystkich żądań skierowanych do obiektu zarządzanego przez POA
  - *osobny wątek dla każdego klienta* osobny proces do obsługi żądań od obiektu stub, który jest klientem zdalnego obiektu zarządzanego przez POA
  - *osobny wątek dla każdego żądania* jeden proces dla każdego zdalnego wywołania metody obiektu zarządzanego przez POA; proces kończy działanie po zakończeniu obsługi zdalnego wywołania

6.4.1 Tłumaczenie zdalnego wywołania metody

Wywołanie metody m z parametrem wejściowym ip i wyjściowym op przez obiekt Stub o nazwie s. Synchroniczne zdalne wywołanie:

```
s.send_sync(m, ip, op)
```

zostanie przekształcone na:

```
s_T.invocation!s_T.stubID, CUP_SYNC, s_T.response, s_T.obj_ID,
               CUP_METHOD_m, ip_T;
stub_T.response?op_T;
```

Żądanie składa się z identyfikatora obiektu stub, flagi typu synchronizacji, kanału odpowiedzi, identyfikatora zdalnego obiektu, identyfikatora zdalnej metody (tutaj system wygenerował CUP_METHOD_m) i parametru wejściowego.

7 Podsumowanie

Weryfikacja prototypu przykładowego systemu aukcyjnego pozwoliła na wykrycie błędów w zachowaniu uczestników licytacji (zakończenie licytacji przed oddaniem głosu) i w synchronizacji serwerów. Nie wykryto blokad w systemie.