Uniwersytet Wrocławski
Instytut Informatyki
Informatyka

Praca magisterska

Dynamiczna analiza poprawności programów na platformę .NET z wykorzystaniem bibliotek do profilowania

Adam Szeliga

Promotor: dr Wiktor Zychla

Wrocław, 2011

Spis treści

Rozdział 1

Wstęp

W niniejszej pracy opisano budowę oraz zasadę działania biblioteki umożliwiającej kontrolę poprawności działania dowolnego programu działającego w obrębie platformy .NET.
Poprawność ta badana jest poprzez weryfikację kontraktów nałożonych na poszczególne części programów, w tym wypadku, funkcji (metod). Ten rodzaj weryfikacji nazywany jest programowaniem kontraktowym. Pojęcie to zostało wprowadzone przez Bertranda Meyera w odniesieniu do języka programowania Eiffel.

Przedstawiona biblioteka została napisana w języku C++ przy wykorzystaniu mechanizmów związanych z technologią COM. Należy wspomnieć, iż w kontekście programowania kontraktowego jest to rozwiązanie, jak do tej pory, unikalne.

1.1 Platforma .NET i CLR

Zasada działania opisywanego rozwiązania mocno opiera się na mechanizmach służących do profilowania aplikacji. Teoria profilowania została przedstawiona w kolejnych rozdziałach. Jak każde podobne rozwiązanie tak i to jest silnie związane ze środowiskiem uruchomieniowym. W tym przypadku jest to platforma .NET stworzona przez firmę Microsoft i przeznaczona dla systemów z rodziny Windows.
Technologia ta nie jest związana z żadnym konkretnym językiem programowania, a programy mogą być pisane w jednym z wielu języków – na przykład C++/CLI, C#, J#, Delphi 8 dla .NET, Visual Basic .NET. Zadaniem kompilatorów jest translacja programów wyrażonych w w/w języków na język pośredni CIL (wcześniej MSIL). Dopiero tak przygotowane programy mogą być wykonane na maszynie wirtualnej CLR, która to jest środowiskiem uruchomieniowym platformy .NET. Taka konstrukcja pozwoliła rozszerzyć zakres działania utworzonego rozwiązania na wszystkie języki

programowania w obrębie tej platformy, pod warunkiem, że dany język wspiera konstrukcje programowe zwane atrybutami.

1.2 Założenia

W celu zapewnienia jak największej użyteczności, zostały przyjęte minimalne założenia co do funkcjonalności jakie muszą być zawarte w bibliotece. Wszystkie z nich zostały szczegółowo opisane w rozdziale czwartym, jednak wprowadzamy je już teraz, aby w dalszej uzasadnić decyzje podjęte przy konstrukcji kolejnych etapów aplikacji.

- biblioteka musi śledzić proces wykonywania programu po jego uruchomieniu
- w celu weryfikacji poprawności programu musi być możliwość zdefiniowana kontraktu
- musi być możliwość odczytania zadanego kontraktu
- aplikacja musi wiedzieć, dla której metody ma się odbyć weryfikacja
- aplikacja musi umieć odczytać argumenty przekazywane do badanych metod
- aplikacja musi zachowywać stan początkowy argumentów metody do momentu jej zakończenia
- aplikacja musi być w stanie odczytać wartości zwracane z badanych metod

W kolejnych rozdziałach opisane w jaki sposób każde z powyższych założeń zostało spełnione. Nie przewidziano żadnych założeń co do wymagań pozafunkcjonalnych, co oznacza, iż takie parametry jak szybkość działania aplikacji czy bezpieczeństwo rozwiązania, nie były przedmiotem zainteresowania.

Rozdział 2

Programowanie kontraktowe

W tym rozdziale została przybliżona specyfika programowania kontraktowego. Programowanie kontraktowe jest metodologią sprawdzania poprawności oprogramowania.

2.1 Historia

Koncepcja ta ma korzenie w pracach nad formalną weryfikacją programów, formalną specyfikacją oraz związanych z logiką Hoara. Wszystkie z powyższych dążą do dowodzenia poprawności programów komputerowych, a tym samym przyczyniają się podnoszeniu ich jakości. Nie inaczej jest w przypadku programowanie kontraktowego. Po raz w obecnej postaci wprowadził je Bertrand Meyer w 1986 roku przy okazji wprowadzenia na rynek projektu języka programowania Eiffel. Do dnia dzisiejszego powstało wiele różnych różnych implementacji tej koncepcji. Cześć języków programowania ma wbudowane mechanizmy pozwalające na definiowanie i sprawdzanie poprawności kontraktów. Do tej grupy zaliczamy:

- Cobra
- Eiffel
- D
- języków opartych na platformie .NET w wersji 4.0

Drugą grupę stanowią języki dla których powstały nakładki umożliwiające ten rodzaj weryfikacji. Ta grupa jest znacznie obszerna i obejmuje większość znaczących języków programowania, takich jak :

- C/C++,

- C#
- Java
- Javascript
- Perl
- Python
- Ruby

Omawiana biblioteka należy do drugiej grupy rozwiązań.

2.2 Opis

Ten rodzaj programowania zakłada, że elementy programu powinny odnosić się do siebie na zasadzie kontraktów, czyli:

- Każdy element powinien zapewniać określoną funkcjonalność i wymagać ściśle określonych środków do wykonania polecenia.
- Klient może użyć funkcjonalności, o ile spełni zdefiniowane wymagania.
- Kontrakt opisuje wymagania stawiane obu stronom.
- Element zapewniający funkcjonalność powinien przewidzieć sytuacje wyjątkowe, a klient powinien je rozpatrzyć.

Koncepcja ta polega na zawieraniu swego rodzaju umowy pomiędzy dostawcą funkcjonalności i klientami. W ogólnym przypadku poprzez dostawców rozumiemy klasy lub metody zawarte w programie, klientem zaś jest każdy kto z tych encji korzysta.
Dla danej klasy kontrakt definiowany jest jako inwariant, to znaczy, warunek jaki musi być spełniony przed i po wywołaniu dowolnej publicznej metody w obrębie tej klasy. Z kolei dla metod kontrakt definiowany przy pomocy warunków początkowego i końcowego, gdzie ten pierwszy specyfikuje jakie założenia powinny być spełnione w momencie wywołania metody, a drugi określa stan aplikacji po jej zakończeniu.

Rozwiązanie, które jest tu opisywane skupia się na drugim rodzaju kontraktów. W języku programowania Eiffel, skąd wywodzi się cała idea, kontrakty opisywane są w sposób następujący:

```
NazwaMetody (deklaracja argumentów) is
require
 -- warunek początkowy
do
 -- ciało metody
ensure
 -- warunek początkowy
end
```

Przy budowie prezentowanej aplikacji wykorzystano cechę szczególną platformy .NET, a w szczególności języka C#, jaką jest możliwość dekorowania metod atrybutami. Ten element języka będzie dokładniej opisany w dalszej części pracy, przy okazji przedstawiania szczegółów implementacyjnych. W tym momencie wystarczy przyjąć, iż atrybuty te stają się częścią meta informacji o danej metodzie, co z kolei może być wykorzystywane przy jej inspekcji. Właśnie ta cecha została wykorzystana w rozważanej aplikacji. Dla ilustracji, poniżej została zademonstrowana ogólna postać zapisu kontraktów w języku C#:

```
[NazwaAtrybutuDefiniującegoKontrakt( warunek początkowy,
                                     warunek końcowy )]
NazwaMetody( deklaracja argumentów )
{
  -- definicja metody
}
```

Rozdział 3

Profilowanie

3.1 Profilowanie – opis

3.2 Typy profilowania

3.2.1 Instrumentacja

3.2.2 Próbkowanie

3.3 Profilowanie w środowisku .NET

3.4 Profilowanie a dynamiczna weryfikacja programów

... jakiś tekst ...

Rozdział 4

Omówienie funkcjonalności biblioteki

W tym rozdziale szczegółowo opisane zostały funkcjonalności jakie udostępnia biblioteka.

4.1 Kontrakty

Podstawowym elementem, dzięki któremu możliwa jest weryfikacja metod, jest oczywiście możliwość definiowania kontraktu. Jak już zostało wspomniane we wcześniejszych rozdziałach kontrakty definiujemy za pomocą atrybutów.

Atrybuty są to znaczniki o charakterze deklaracyjnym zawierające informację o elementach programu (np. klasach, typach wyliczeniowych, metodach) przeznaczoną dla środowiska wykonania programu. Co jest w tym kontekście istotne to iż są one pamiętane jako meta-dane elementu programu.

Definicja atrybutów jest jedynym elementem, wchodzącym bezpośrednio w skład omawianego rozwiązania, który musi znajdować się po stronie weryfikowanej aplikacji.

4.2 Inspekcja nadzorowanego programu

Jak to zostało wspomniane we wcześniejszych rozdziałach, aplikacja weryfikująca kontrakty ma postać biblioteki COM i jako taka musi być wcześniej zarejestrowana w systemie. Do tego celu używana jest aplikacja o nazwie regsrv32.exe, która to jest częścią narzędzi dostarczanych wraz z platformą .NET. Zadaniem tego narzędzia jest pobranie identyfikatora biblioteki i umieszczenie w rejestrze systemu klucza przechowującego ten identyfikator oraz ścieżkę w systemie pliku pod która znajduje się

biblioteka.

Rozpoczęcie procesu profilowania/weryfikacji aplikacji odbywa się poprzez uruchomienie programu z linii poleceń w odpowiednio przygotowanym środowisku. Etap ten polega na ustawieniu zmiennych środowiskowych, instruujących maszynę wirtualną CLR, aby ta wysyłała powiadomienia na temat zdarzeń zachodzących wewnątrz uruchamianej aplikacji. Proces ten wygląda w sposób następujący:

SET COR_ENABLE_PROFILING=1
SET COR_PROFILER={GUID}

Powyższe zmienne są następnie odczytywane przez środowisko uruchomieniowe. Pierwsza z nich informuje maszynę wirtualną, że ta powinna przesyłać informacje o zdarzeniach do biblioteki, której położenie określanie jest przy wykorzystaniu identyfikatora GUID.

Liczba i rodzaj wysyłanych powiadomień określany jest wewnątrz biblioteki profilującej. W szczegółach temat ten opisany jest w kolejnym rozdziale.

4.3 Odczyt i interpretacja metadanych

4.4 Odczytywanie wartości obiektów

4.5 Wartości początkowe

4.6 Wartości zwracane

4.7 Ograniczenia

Rozdział 5

Szczegóły implementacji

Rozdział 6

Porównanie z innymi bibliotekami

6.1 CodeContracts

6.2 LinFu.Contracts

Rozdział 7

Podsumowanie

Bibliografia