Specyfikacja do projektu

"Diagramy UML"

realizowanego w ramach przedmiotu Studio Projektowe 1.

Autorzy:

Dominik Szczygieł

Sebastian Hoffman

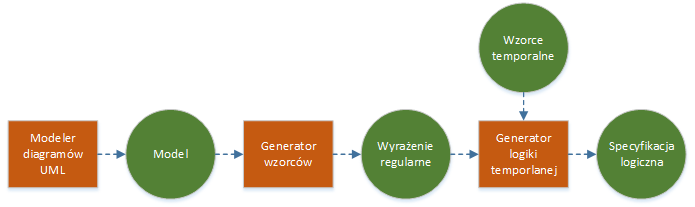
Wersja 1

7 września 2015 roku

# Cel aplikacji

Zadaniem aplikacji jest generowanie specyfikacji logicznej na podstawie analizy podanego przez użytkownika diagramu aktywności UML.

# Schemat działania aplikacji



# Modeler

Zadaniem modelera jest możliwość utworzenia diagramu aktywności, który zostaje poddany analizie. Oprócz łatwości tworzenia takiego diagramu, program powinien być dostępny na bezpłatnej licencji oraz umożliwiać eksportowanie utworzonego modelu do formatu umożliwiającego łatwe przetwarzanie.

Spośród dostępnych na rynku aplikacji został wybrany **Visual Paradigim**. Aplikacja jest bardzo prosta w obsłudze a ponadto umożliwia eksportowanie diagramów aktywności w formacie XML.

# Generator wzorców

Generator przetwarza diagram aktywności oraz wyszukuje w nim określone wzorce. Wynikiem działanie jest wyrażenie regularne.

Poprawie zbudowany diagram aktywności daje się przedstawić jako wzorzec, składający się z pojedynczych węzłów lub też innych wzorców. Nie jest to możliwe tylko w przypadku, kiedy diagram składa się z innych wzorców niż przewidziane.

Wzorce obsługiwane przez aplikację:

1. **Sekwencja –** Seq(a,b)



1. **Podwójna sekwencja** – SeqSeq(c,d,e)



1. **Zrównoleglenie** – Par(a,b,c)

****

1. **Rozgałęzienie**  - Dec(dec1,c,b)

****

1. **Pętla** – Loop(d, dec2, e)

****

# Generatorlogiki temporalnej

Zadaniem tego generatora jest analiza wyrażenia regularnego z wykorzystaniem podanego algorytmu przetwarzania i pliku zawierającego opis logiczny każdego z wyspecyfikowanych wzorców. Wynikiem jego działania jest specyfikacja logiczna.

**Algorytm działania generatora**:

1. Na początku zbiór formuł wynikowych jest pusty (*L=ø*). Na wejściu znajduje się wyrażenie *WL* składające się z predefiniowanych wzorców i będące wynikiem działania generatora wzorców.
2. Wyrażenie *WL* analizowane jest od lewej strony do prawej.
3. Jeśli analizowany wzorzec (*wrf*) zawiera wyłącznie formuły atomowe (nie zawiera w sobie zagnieżdżonych innych wzorców), to specyfikacja logiczna jest rozszerzana o zbiór formuł związanych z rodzajem analizowanego wzorca (*L := L* ∪ *wrf()T*)
4. Jeżeli którykolwiek z argumentów (*r()*) jest zagnieżdżonym wzorcem, to specyfikacja logiczna jest rozszerzana o zbiór formuł powstałych na skutek wstawienia w miejscu tego argumentu alternatywy jego wyrażenia wejściowego - *ini* (re) i wyjściowego - *fin (rx)*, (*L := L* ∪ ((*wrf()T* ) ← “*r()e* ∨ *r()x*”)).

# Ograniczenia działania

Przed rozpoczęciem używania programu, należy mieć na uwadze pewne ograniczenia wynikające z realizacji logiki aplikacji. Zostaną one wymienione w punktach:

1. Aplikacja posiada domyślną konfigurację obsługiwanych wzorców – tj. domyślnie każdy wzorzec posiada przypisany priorytet wyszukiwania w drzewie aktywności:

|  |  |  |
| --- | --- | --- |
| **Wzorzec** | **Priorytet** | **Ilość węzłów składowych** |
| LOOP | 5 | 3 |
| PAR | 4 | 3 |
| DEC | 3 | 3 |
| SEQSEQ | 2 | 3 |
| SEQ | 1 | 2 |

Bez wczytywania pliku konfiguracyjnego program będzie działał poprawnie. Użytkownik ma możliwość zmiany priorytetów za pomocą zewnętrznego pliku konfiguracyjnego. Przykładowa zawartość pliku:

SEQ:2  
SEQSEQ:1  
PAR:3  
DEC:5  
LOOP:4

[NAZWA\_WZORCA:PRIORYTET]

W przypadku użycia zewnętrznej konfiguracji, w pliku **muszą** pojawić się wszystkie obsługiwane wzorce wraz z przypisanymi priorytetami. Jeżeli w pliku konfiguracyjnym znajdzie się nieobsługiwany wzorzec lub ilość wzorców nie będzie kompletna, aplikacji poinformuje użytkownika o błędzie, pozostając w domyślnej konfiguracji. **Aby aplikacja obsługiwała nowe wzorce konieczna jest ingerencja w kod źródłowy !** Wiąże się to przede wszystkim z koniecznością implementacji mechanizmu pozwalającego wykryć nowy wzorzec w diagramie aktywności.

1. Nie jest możliwe rozpoczęcie działania aplikacji bez wczytania pliku z specyfikacją LTL każdego ze wzorców. Plik ten musi posiadać ściśle określoną strukturę. Specyfikacje dla kolejnych wzorców muszą być rozdzielone linią ze znakiem „@”.

Celem przedstawienia schematu zapisu, zostanie przedstawiona specyfikacja dla pojedynczego wzorca:

Seq(f1,f2)

ini= f1 / fin= f2

<>f1 / <>f2

f1 => <>f2

f2 => ~<>f2

Pierwsza linia musi zawierać nazwę wzorca ( wielkość liter nie ma znaczenia ) wraz z podanymi w nawiasach, zaraz przy nazwie, schematowymi nazwami węzłów składowych. **Ilość węzłów składowych, musi być równa, zapisanej w kodzie programu ilości, przedstawionej w tabeli w punkcie 1.**

Druga linia zawiera informacje o węzłach początkowych ( ini ) oraz węzłach końcowych ( fin ) danego wzorca. Obowiązuje zapis jak w przykładzie.

Kolejne linie aż do napotkania znaku „@” lub końca pliku są specyfikacją LTL danego wzorca.

Niezgodność ze strukturą skutkuje błędem, o którym użytkownik zostaje powiadomiony komunikatem. Podobnie jak w przypadku pliku konfiguracyjnego, plik ze specyfikacją **musi zawierać logikę dla każdego obsługiwanego wzorca.** W przypadku niekompletności lub wykrycia nieobsługiwanego wzorca nie jest możliwa generacja logiki.

1. Przetwarzanie diagramu aktywności możliwe jest dopiero po wczytaniu pliku ze specyfikacją LTL dla wszystkich wzorców. W przypadku błędu podczas ładowania diagramu aktywności, użytkownik zostanie o tym poinformowany odpowiednim komunikatem.

Zarówno generacja wyrażenia regularnego jak i specyfikacji logicznej wymaga pełnej poprawności podanego przez użytkownika diagramu aktywności, tj.:

* Diagram aktywności **musi** posiadać element początkowy, oznaczany jako INITIAL NODE
* Struktura diagramu składa się z takiej kombinacji wzorców, która daje się uprościć do jednego wzorca, mogącego zawierać zarówno inne wzorce jak i pojedyncze węzły

Drugie założenie oznacza, że w diagramie aktywności, nie może pojawić się sekwencja węzłów, tworząca wzorzec nierozpoznawalny przez aplikację. Również na etapie przetwarzania diagramu jak i generacji logiki użytkownik jest powiadamiany o błędach.

# Architektura aplikacji

Aplikacja została stworzona jako web serwis. Do jej realizacji został użyty język Java w wersji 7 wraz z webowym framework’iem Spring w wersji 4.1.6. Dodatkowe używane biblioteki funkcjonalne to Google Guava oraz log4j. Kod aplikacji kompiluje się do wykonywalnego pliku jar. Po jego uruchomieniu na maszynie zostaje uruchomiony serwer http odpowiadający na zapytania serwisu. Komunikacja pomiędzy serwerem a przeglądarką odbywa się w stylu REST. Do poprawnego działania aplikacji wymagany jest zainstalowane środowisko uruchomieniowe języka Java ( JRE ) w wersji przynajmniej 1.7.

# Uruchomienie aplikacji

Aby rozpocząć korzystanie z aplikacji, należy uruchomić serwis odpowiadający za obsługę zapytań HTTP. Można to zrobić w dwojaki sposób – uruchamiając plik wykonywalny jar bezpośrednio z folderu lub poprzez wiersz poleceń. Drugi sposób umożliwia wygodne zakończenie działania aplikacji ponieważ, wystarczy zamknąć okno wiersza poleceń aby zabić odpowiadający aplikacji proces. Pierwszy sposób wiąże się z manualnym zabiciem procesu o nazwie javaw.

Aby uruchomić aplikację z poziomu wiersza poleceń, należy udać się do folderu w którym znajduje się paczka i wydać komendę:

*java –jar DiagramyUML.jar*