Specyfikacja do projektu "Diagramy UML" realizowanego w ramach przedmiotu Studio Projektowe 2.

Autorzy: Agnieszka Biernacka, Jerzy Biernacki

Opiekun pracy: dr inż. Radosław Klimek

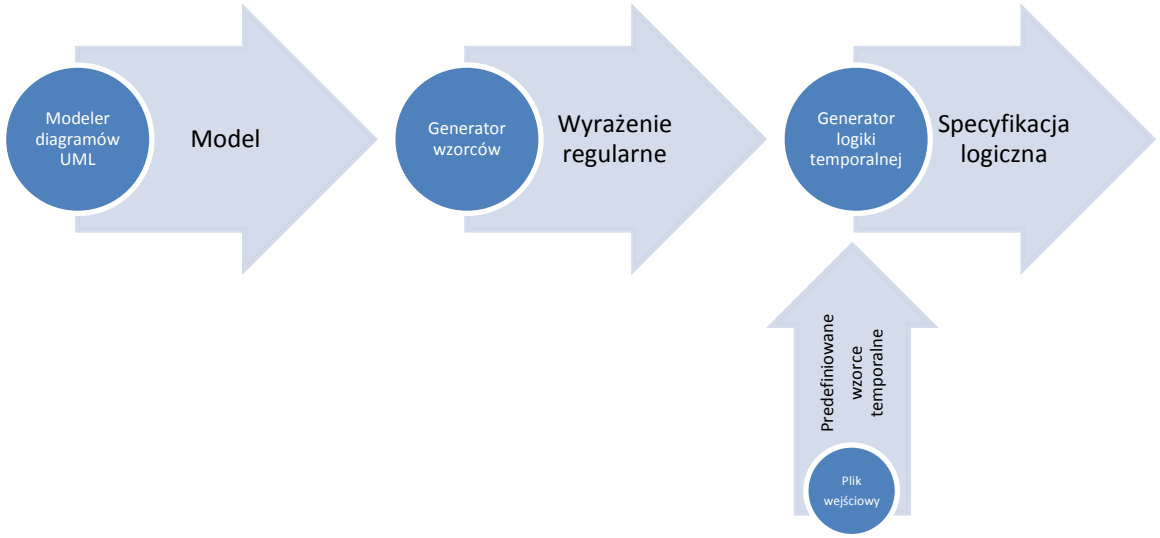
Wersja nr 8

12 stycznia 2014 roku

1. Cel aplikacji.

Zadaniem programu jest generacja specyfikacji logicznej na podstawie analizy danych pobranych z określonego modelera diagramów aktywności UML.

2. Schemat działania aplikacji.



3. Opis poszczególnych części programu.

3.1. Modeler

Zadaniem modelera jest dostarczenie modelu diagramu aktywności do późniejszej analizy. Wynikiem działania modelera powinien być łatwy do odczytu i przetworzenia zapis modelu.

Program powinien spełniać następujące kryteria:

- powinien być udostępniony na bezpłatnej licencji
- powinien udostępniać zaawansowane możliwości tworzenia i edycji diagramów aktywności UML
- powinien udostępniać możliwość exportu diagramu do łatwego do przetworzenia formatu zapisu danych, np. XML.

W ramach pierwszej fazy projektu dokonano analizy istniejących narzędzi tworzenia diagramów UML. Pod uwagę brane były następujące programy:

- Enterprise Architect
- Umlet
- Altova
- Visual Paradigim

Wszystkie z nich udostępniały export do formatu UML. Z podanej czwórki jedynie Umlet i Visual Paradigim udostępniały bezpłatną licencję, jednak wybrany został program Visual Paradigim, ze względu na jego znacznie większe możliwości.

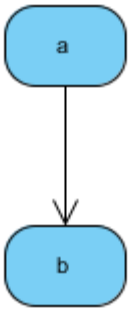
3.2 Generator wzorców.

Zadaniem generatora jest analiza modelu diagramu UML polegająca na wyszukiwaniu w nim określonych z góry wzorców. Wynikiem jego działania jest wyrażenie regularne.

Założeniem programu jest, że podany model składa się tylko i wyłącznie z podanych wzorców. Jeśli podczas analizy generator napotka elementy nie pasujące do wzorców, program powinien zakończyć się niepowodzeniem informując użytkownika o błędzie.

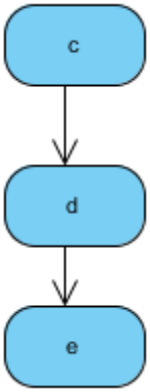
Wzorce:

A. Sekwencja - posiada najniższy priorytet (4)



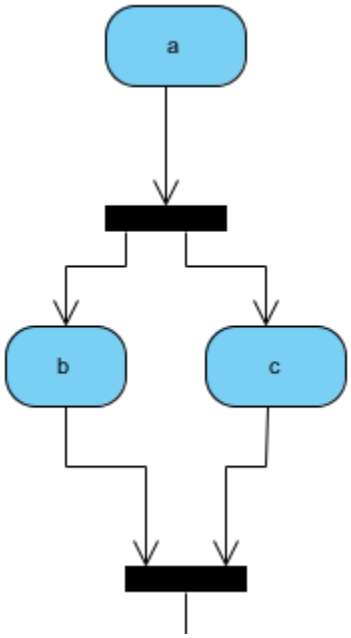
Seq(a,b)

B. Podwójna sekwencja - posiada niski priorytet (3)



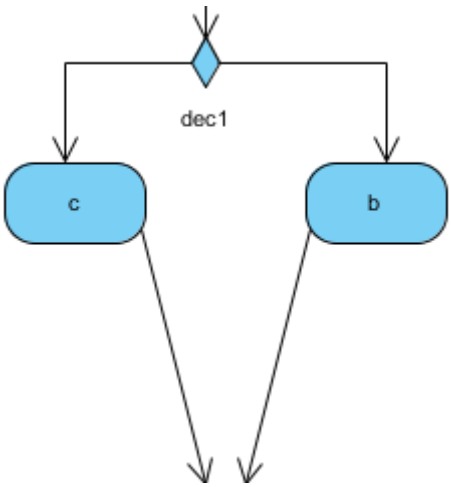
Seqseq(c,d,e)

C. Zrównoleglenie - posiada średni priorytet (2)



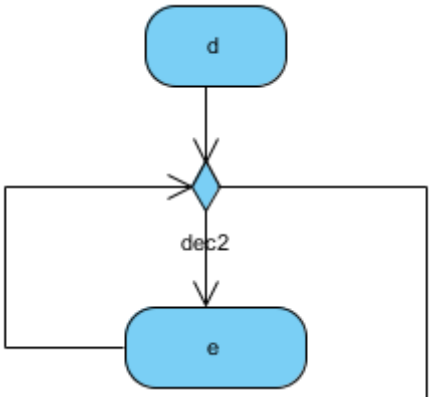
Par(a,b,c)

D. Rozgałęzienie - posiada średni priorytet (2)



Dec(dec1,c,b)

E. Pętla - posiada najwyższy priorytet (1)



Loop(d,dec2,e)

W programie założone jest, że kolejność zapisu wzorców nie ma znaczenia, np. seq(a,seq(b,c)) <=> seq(seq(a,b),c).

3.3 Generator logiki 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 wygląda następująco:

1. Na początku zbiór formuł wynikowych jest pusty ($L=\emptyset$). Na wejściu znajduje się wyrażenie W_L składające się z predefiniowanych wzorców i będące wynikiem działania generatora wzorców.

2. Wyrażenie W_L 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 ($L := L \cup wrf()^T$)

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* (r^e) i wyjściowego - *fin* (r^x), ($L := L \cup ((wrf()^T) \leftarrow "r()^e \vee r()^x")$).

.

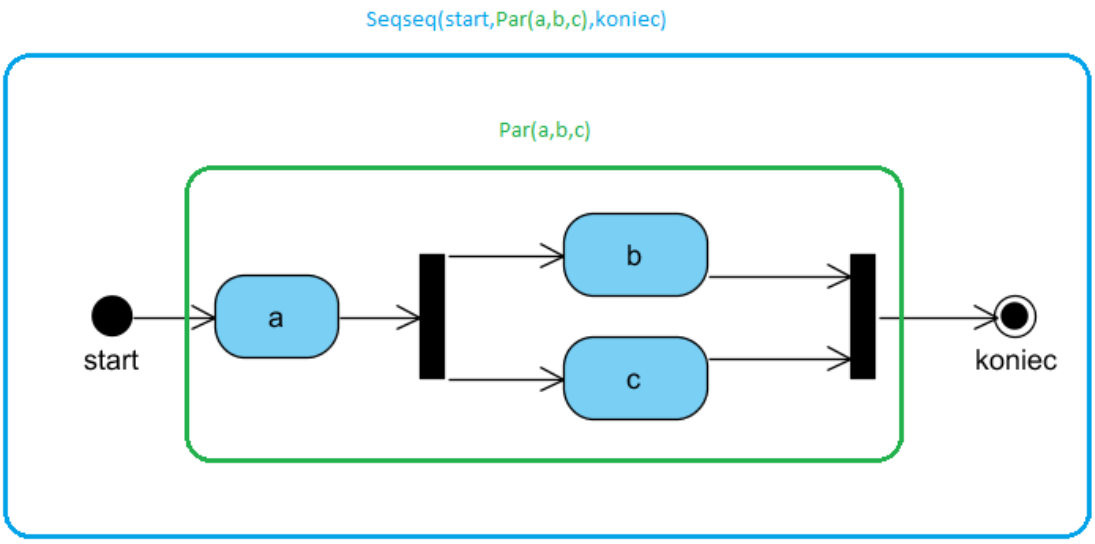
4. Analiza działania programu na przykładach

W tym rozdziale znajdą się analizy przykładów działania programu dla 5 różnych diagramów aktywności. W przykładach posłużyliśmy się przykładowym plikiem zawierającym opis reguł LTL dla wzorców. Plik ten nie jest w pełni poprawny pod względem reguł LTL - jest jedynie testem działania algorytmu. Dla ułatwienia analizy zastosowaliśmy kolorowanie składni. Plik wygląda następująco:

```
Seq(f1,f2):
ini= f1 / fin= f2
<>f1 / <>f2
f1 => <>f2
f2 => ~<>f2
SeqSeq(f1,f2,f3):
ini= f1 / fin= f3
<>f1 / <>f2 / <>f3
f1 => <>f2
f2 => ~<>f2
f2 => <>f3
f3 => ~<>f3
Par(f1,f2,f3):
ini= f1 / fin= f2 & f3
<>f1 / <>f2 / <>f3
f1 => <>f2 & <>f3
(f2 | f3) => ~<>f1
[]~(f1 & (f2 | f3))
Dec(f1,f2,f3):
ini= f1 / fin= (f2 & ~f3) | (~f2 & f3)
<>f1 / <>f2 | <>f3
f1 => ((<>f2 & ~<>f3) | (~<>f2 & <>f3))
f2 | f3 => ~<>f1
Loop(f1,f2,f3):
ini= f1 / fin= f2
<>f1 / <>f2
f1 => <>f2
f3 => <>f2
(f2 | f3) => ~<>f1
```

Pliki zawierające zbiór formuł również zostały pokolorowane dla ułatwienia odczytu. Dodatkowo w zbiorze formuł nie dokonywano usunięcia zduplikowanych formuł. Dla zwiększenia przejrzystości, każda formuły wygenerowane z wzorców analizowanych według algorytmu są oddzielone pustymi liniami.

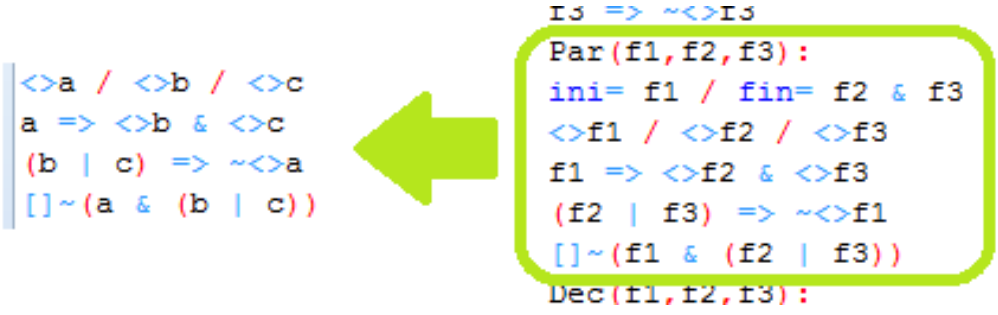
4.1 Dokładna analiza prostego przypadku.



Wykryty wzorzec:

`seqseq(start,par(a,b,c),koniec)`

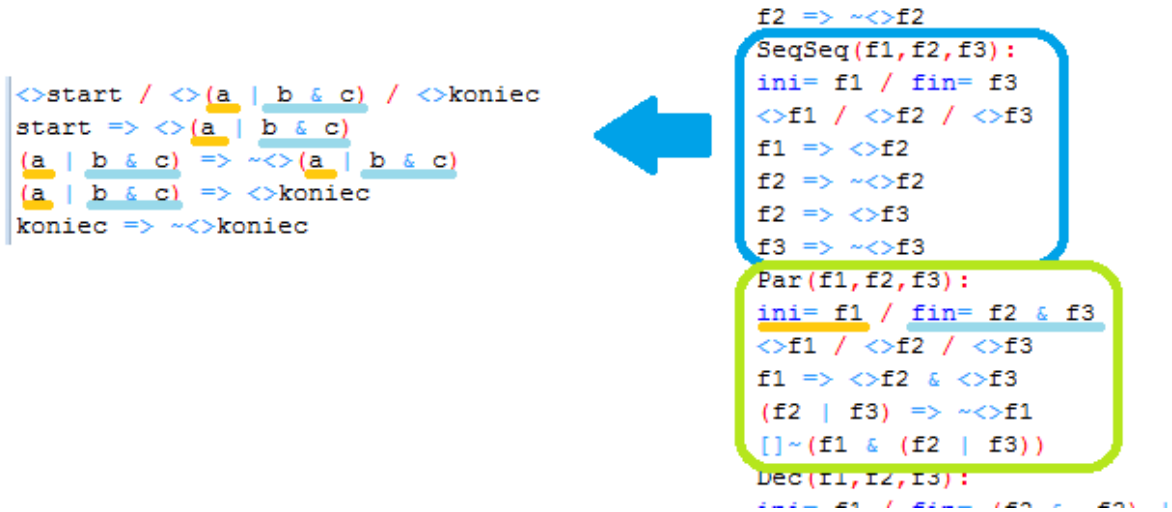
A. Rozpoczynamy od najprostszego wzorca, a więc **par**.
Nie posiada on w sobie zagnieżdżonych żadnych innych wzorców, więc do zbioru formuł dopisywane są formuły z podanego pliku i odpowiednie argumenty podstawiane są w miejsca parametrów f1,f2 i f3:



B. Kolejnym wzorcem jest **Seqseq**.
Jednym z argumentów wzorca **Seqseq** jest w tym wypadku inny wzorzec (**par**).

Tak więc najpierw do zbioru formuł dopisujemy formuły dla **Seqseq** w miejscu argumentu f2 podstawiając alternatywę wyrażeń ***ini*** oraz ***fin*** wzorca **par**, które są w tym wypadku odpowiednio: ***a*** oraz *b & c* . Argumenty f1 i f3 wynoszą odpowiednio **start** i **koniec**.

Formuły dla tego wzorca zostaną stworzone w następujący sposób:

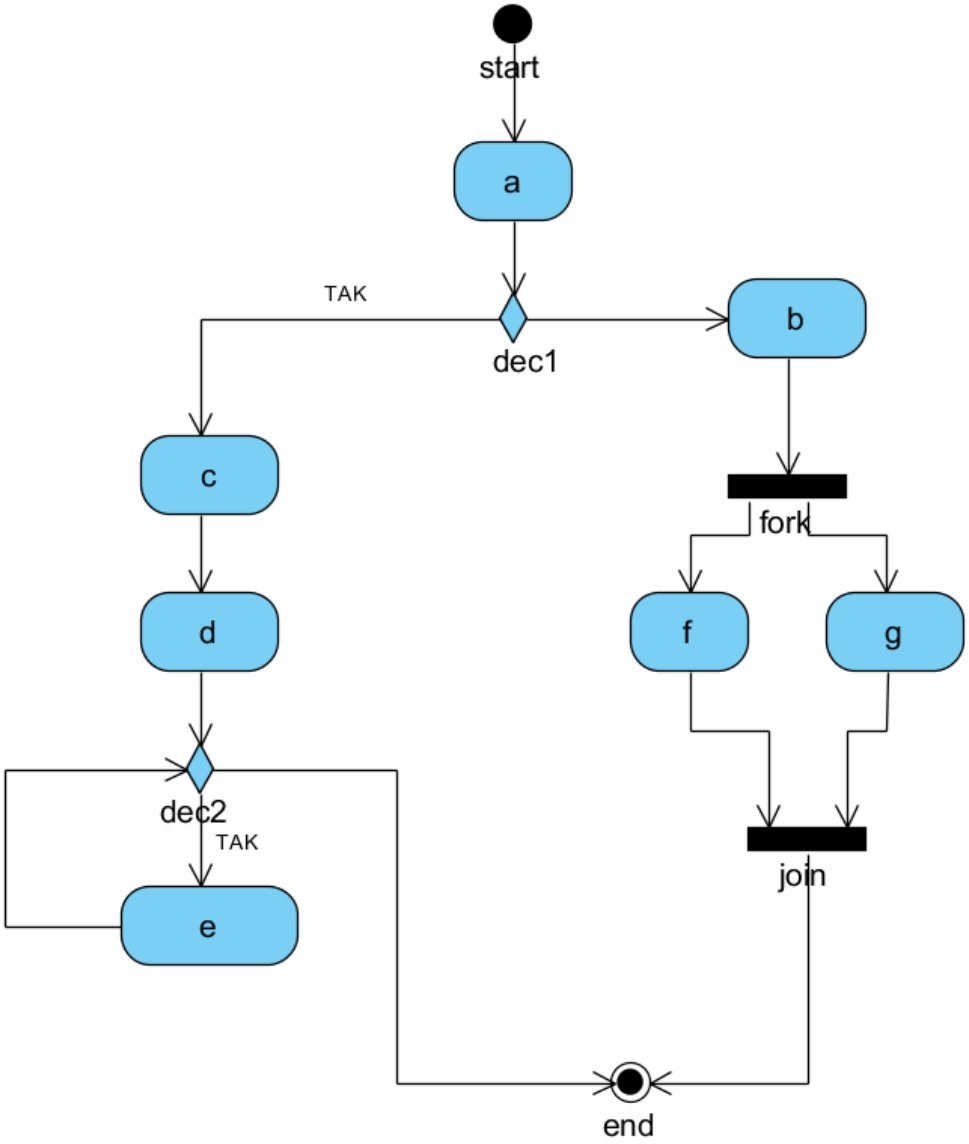


Całość przedstawia się następująco:

```
<>a / <>b / <>c
a => <>b & <>c
(b | c) => ~<>a
[]~(a & (b | c))
```

```
<>start / <>(a | b & c) / <>koniec
start => <>(a | b & c)
(a | b & c) => ~<>(a | b & c)
(a | b & c) => <>koniec
koniec => ~<>koniec
```

4.2 Przykład 2 – proste wykorzystanie wszystkich wzorców.



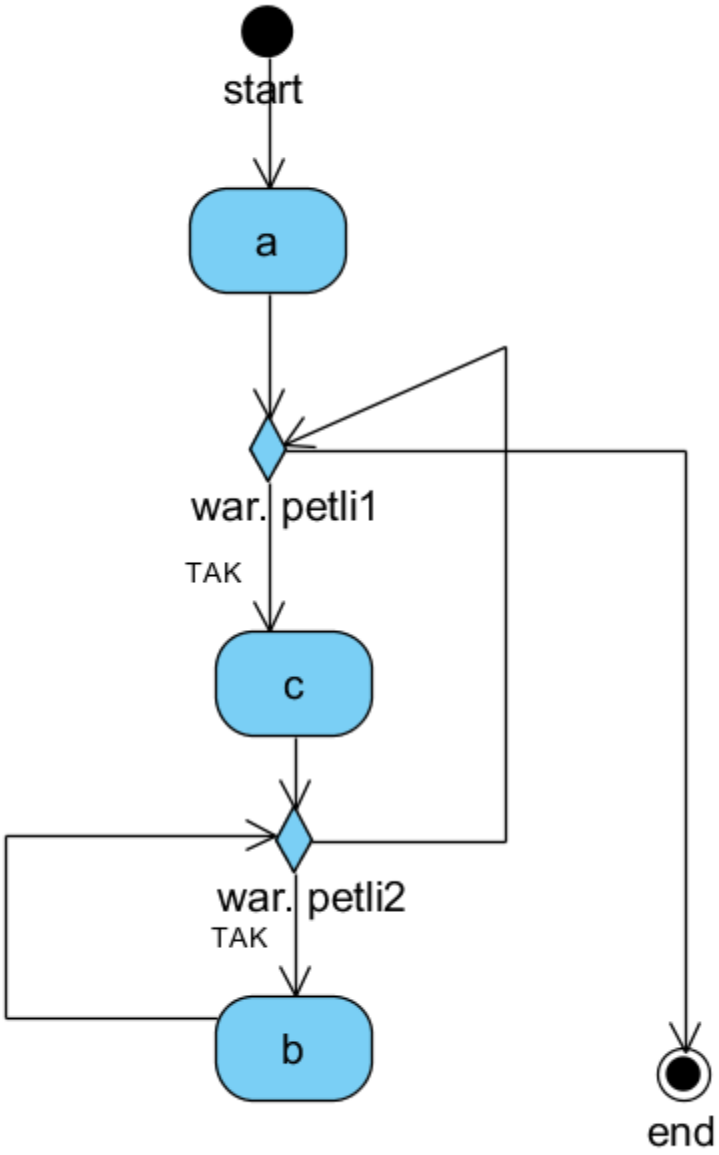
Wykryty wzorzec:

```
seq(start,seqseq(a,dec(dec1,seq(c,loop(d,dec2,e)),par(b,f,g)),end))
```

Zbiór formuł LTL:

```
<>d / <>dec2
d => <>dec2
e => <>dec2
(dec2 | e) => ~<>d

<>c / <>(d | dec2)
c => <>(d | dec2)
(d | dec2) => ~<>(d | dec2)

<>b / <>f / <>g
b => <>f & <>g
(f | g) => ~<>b
[]~(b & (f | g))

<>dec1 / <>(c | dec2) | <>(b | f & g)
dec1 => ((<>(c | dec2) & ~<>(b | f & g)) | (~<>(c | dec2) & <>(b | f & g)))
(c | dec2) | (b | f & g) => ~<>dec1

<>a / <>(dec1 | ((dec2 & ~f  &  g) | (~dec2 & f  &  g))) / <>end
a => <>(dec1 | ((dec2 & ~f  &  g) | (~dec2 & f  &  g)))
(dec1 | ((dec2 & ~f  &  g) | (~dec2 & f  &  g))) => ~<>(dec1 | ((dec2 & ~f  &  g) | (~dec2 & f  &  g)))
(dec1 | ((dec2 & ~f  &  g) | (~dec2 & f  &  g))) => <>end
end => ~<>end

<>start / <>(a | end)
start => <>(a | end)
(a | end) => ~<>(a | end)
```

4.3 Przykład 3 - podwójna pętla:



Wykryty wzorzec:

```
seqseq(start,loop(a,war. petli1,loop(c,war. petli2,b)),end)
```

Zbiór formuł LTL:

```
<>c / <>war. petli2
c => <>war. petli2
b => <>war. petli2
(war. petli2 | b) => ~<>c

<>a / <>war. petli1
a => <>war. petli1
(c | war. petli2) => <>war. petli1
(war. petli1 | (c | war. petli2)) => ~<>a

<>start / <>(a | war. petli1) / <>end
start => <>(a | war. petli1)
(a | war. petli1) => ~<>(a | war. petli1)
(a | war. petli1) => <>end
end => ~<>end
```

5. Instrukcja obsługi aplikacji

Aplikacja implementująca zadany problem projektowy nazwana została *Uml2Ltl*.

Cała logika aplikacji napisana została w języku Java i jest możliwa do uruchomienia w trybie konsolowym na dowolnej platformie obsługującej wirtualną maszynę Javy (JVM). Jej działania opisane zostanie w rozdziale 5.1.

Dla ułatwienia obsługi aplikacji została napisana nakładka graficzna do aplikacji w języku C++, przeznaczona na platformę Windows. Opis jej użycia został zawarty w rozdziale 5.2.

5.1 Instrukcja obsługi aplikacji konsolowej

1. Aplikację konsolowa włącza się uruchamiając w konsoli plik uml2ltl.jar :

   $ java -jar uml2ltl.jar

Dostępne opcje:

--gui : Aplikacja kontrolowana przez GUI, tryb interaktywny - nie do użytku w standardowym cmd

-c <Ścieżka pliku> : Plik wejściowy zawierający opis wzorców w LTL

-i <Ścieżka pliku>: Plik wejściowy modelu UML zapisanego w XML

-o <Ścieżka pliku>: Plik wynikowy zawierający wygenerowane formuły LTL

Opcje **-c** , **-o** oraz **-i** są obowiązkowe.

2. Jeśli podane zostaną złe opcje aplikacja wyświetli pomoc.
3. Plik wynikowy z formułami LTL generuje sie do pliku podanego w argumencie *-o* albo w sta

5.2 Nakładka graficzna

Nakładkę graficzną włącza się uruchamiając plik *Uml2Ltl GUI.exe.* Aplikacja składa się z trzech kart (zakładek), pomiędzy którymi przemieszczać można się za pomocą przycisków *Wstecz* i *Dalej* oraz klikając na odpowiedniej zakładce karty.

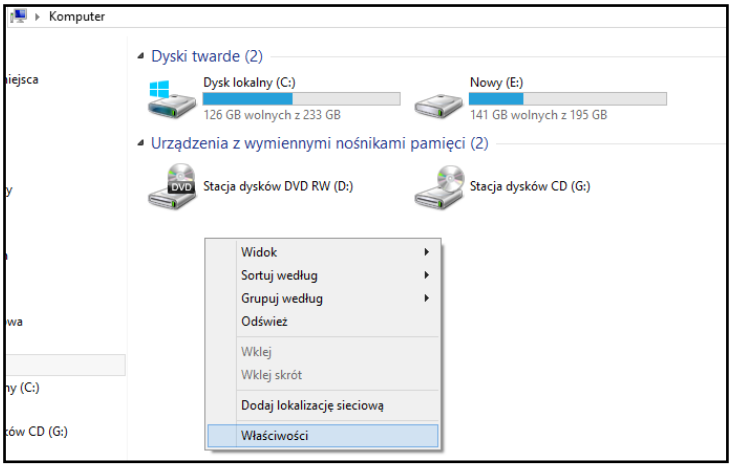
Ważne informacje:

- Aplikacja konsolowa (plik *uml2ltl.jar*) musi znajdować się w tej samej lokalizacji co nakładka
- Plik konfiguracyjny (*conf.xml*) musi znajdować się w tej samej lokalizacji co nakładka
- Jeśli generacja LTL zatrzymuje się podczas łączenia z aplikacją konsolową, oznacza to, że należy dodać Javę do zmiennych środowiskowych.

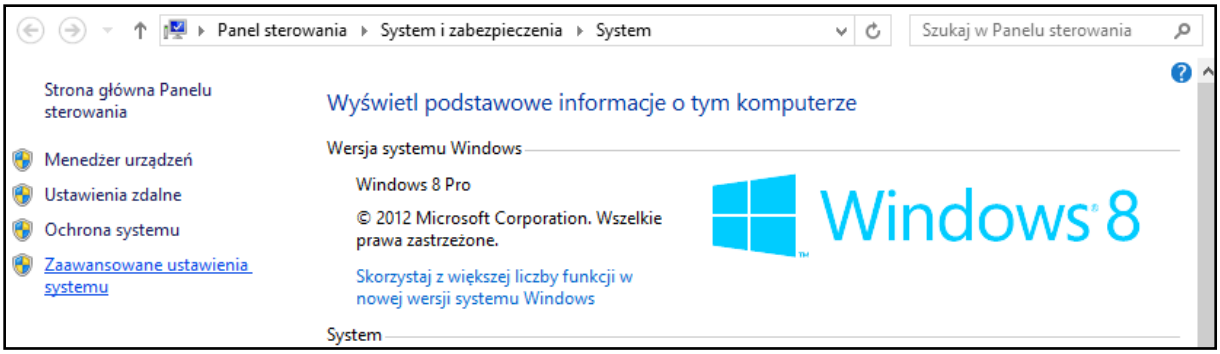
Ustawienie zmiennej środowiskowej Javy

Windows 7/8:

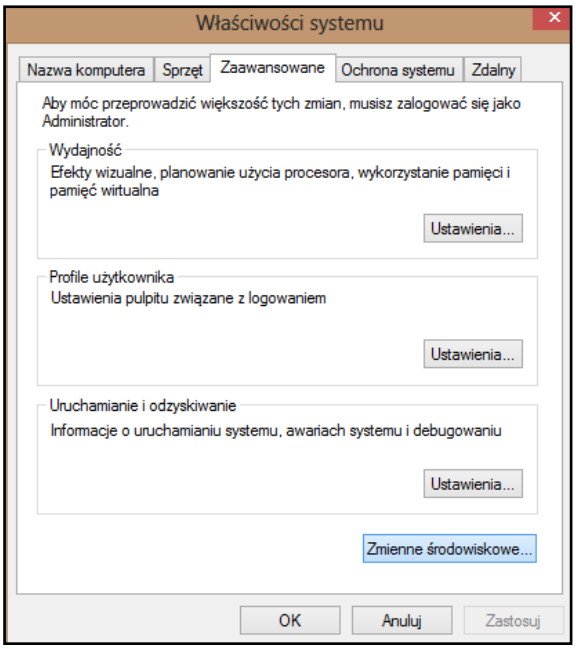
Wchodzimy we właściwości komputera:



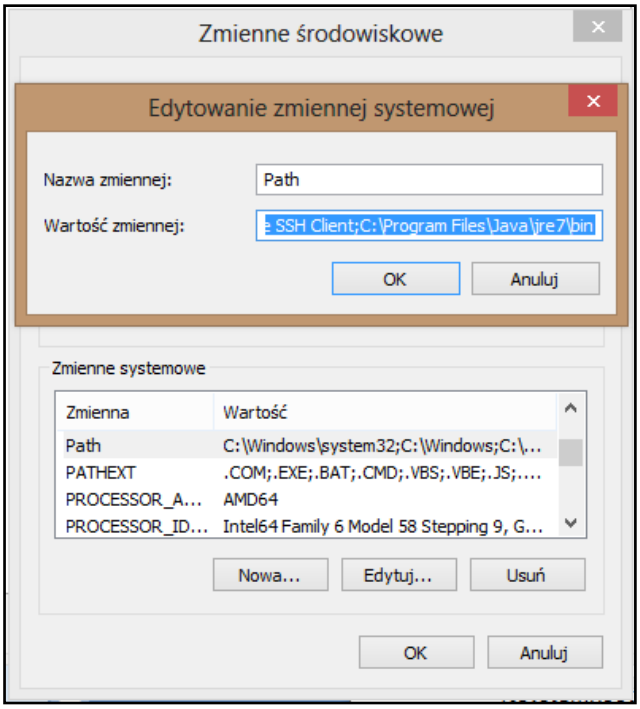
Następnie wybieramy *Zaawansowane ustawienia systemu*:



Dalej klikamy przycisk *Zmienne środowiskowe*:



Edytujemy zmienną Path znajdującą się w zmiennych systemowych dopisując na końcu ścieżkę do plików wykonywalnych Javy, np. "C:\Program Files\Java\jre7\bin"



Przykładowo dla systemu z którego korzystamy ścieżka Path po edycji wygląda następująco:

%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Program Files\TortoiseSVN\bin; C:\Program Files\Java\jre7\bin

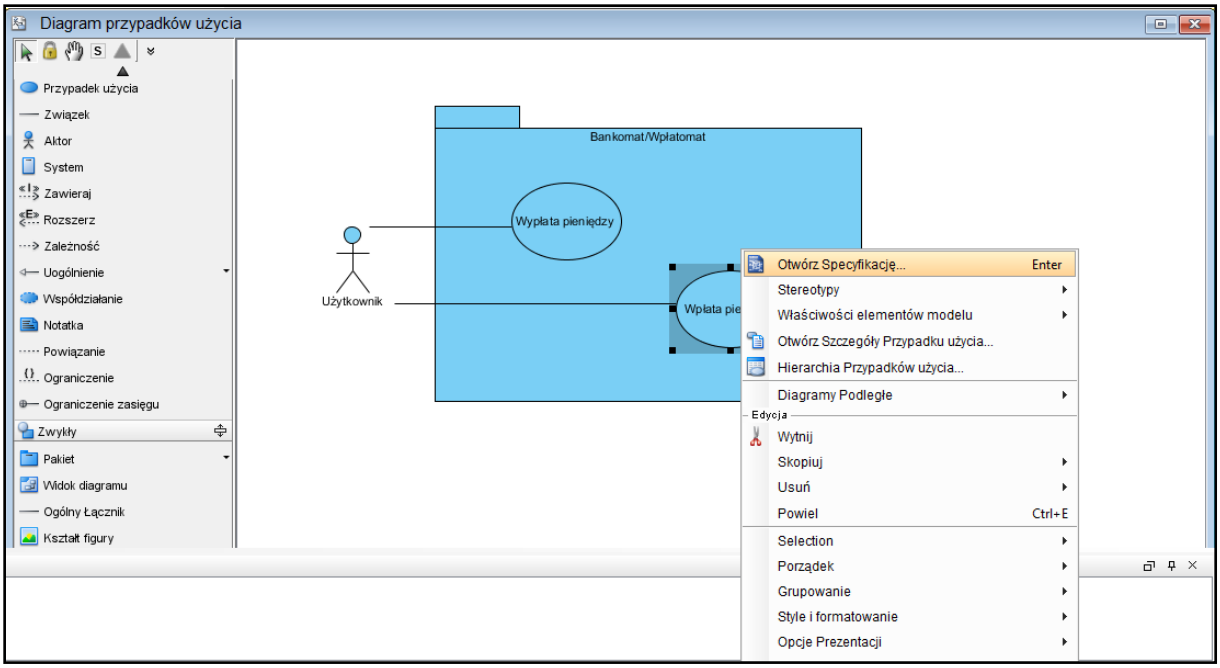
Informacje o ustawieniu zmiennej środowiskowej Path dla Javy na innych systemach operacyjnych można znaleźć pod adresem: http://www.java.com/pl/download/help/path.xml.

5.3 Przykład użycia aplikacji krok po kroku

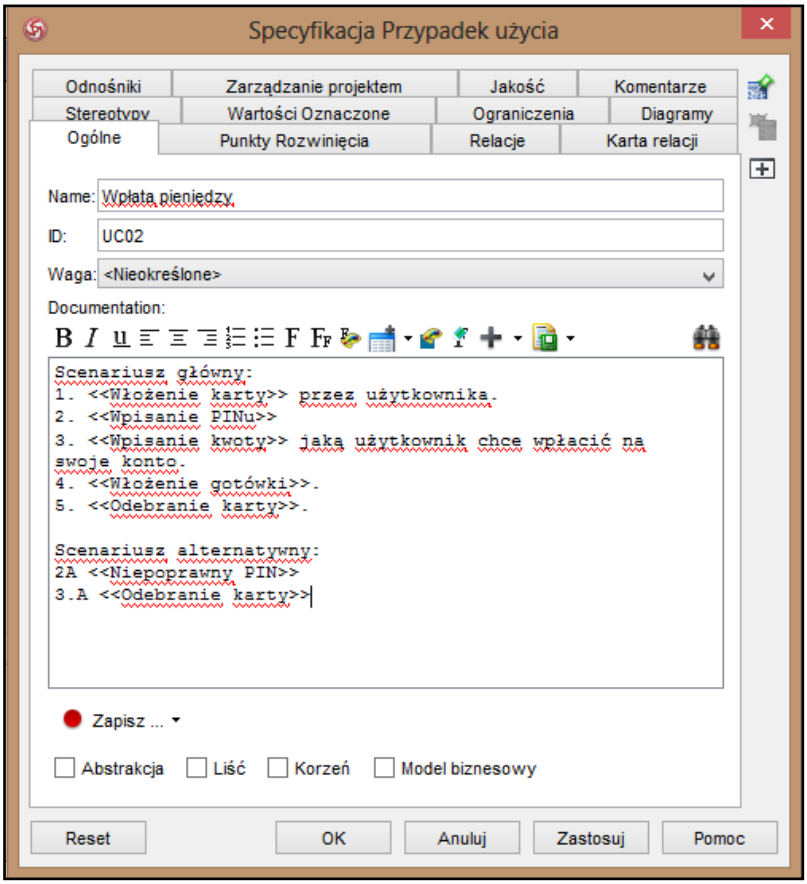
1. Utworzenie diagramu przypadków użycia przy pomocy Visual Paradigm

W Visual Paradigm tworzymy diagram przypadków użycia. Przykładowo:

Aby dodać scenariusz dla przypadku użycia, klikamy prawym klawiszem myszy na wybrany przypadek użycia, po czym wybieramy opcję "Otwórz specyfikację":



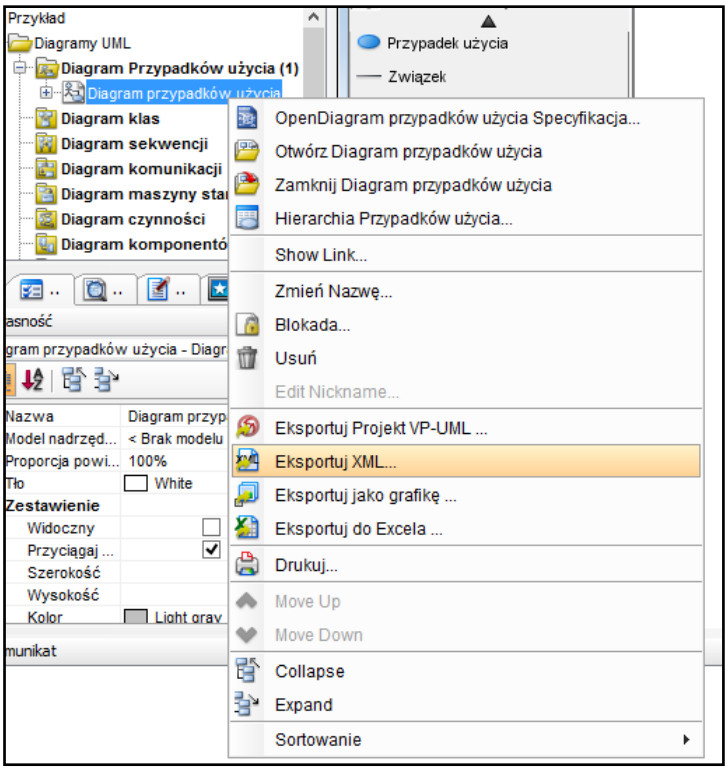
W zakładce "Ogólne" zapisujemy scenariusz przypadku użycia:



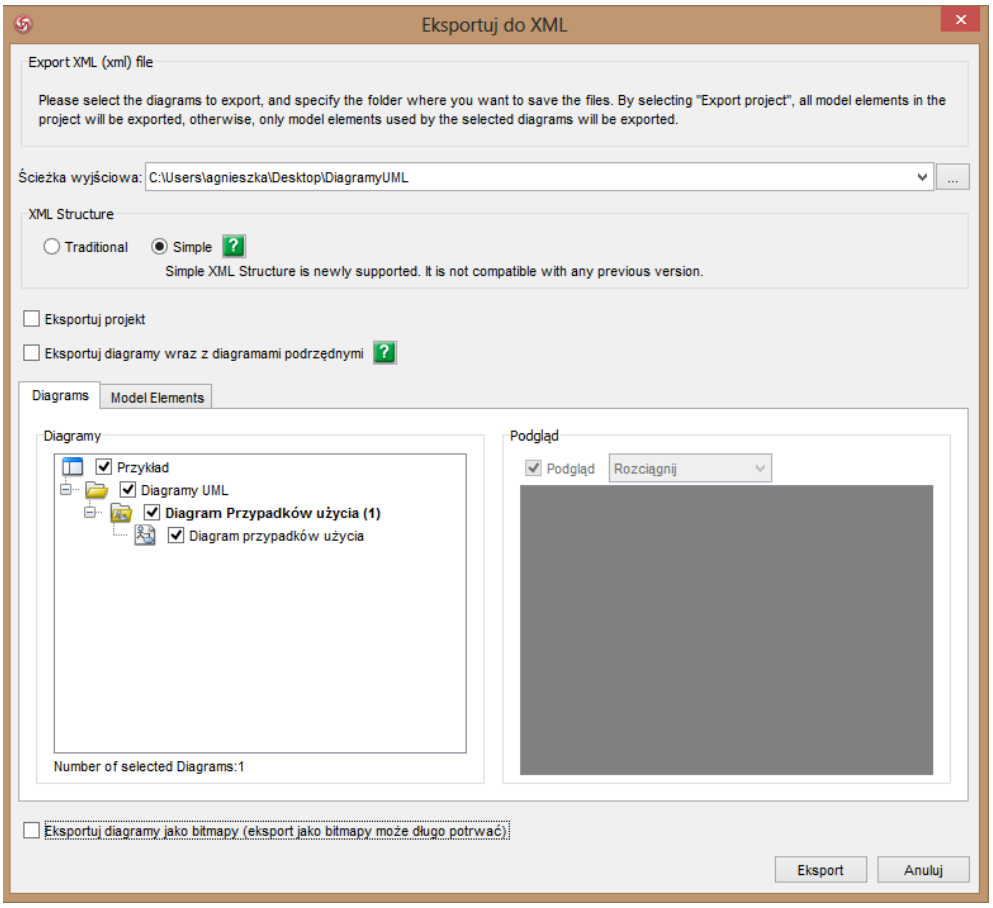
Scenariusz powinien być utworzony dla każdego z przypadków testowych. Słowa/frazy jakie chcemy aby należały do słownika wstawiamy w znaki <<*słowo*>>. W przypadku przedstawionym powyżej do słownika trafi:

- *Włożenie karty*
- *Wpisanie PINu*
- *Wpisanie kwoty*
- *Włożenie gotówki*
- *Odebranie karty*
- *Niepoprawny PIN*
- *Odebranie karty*

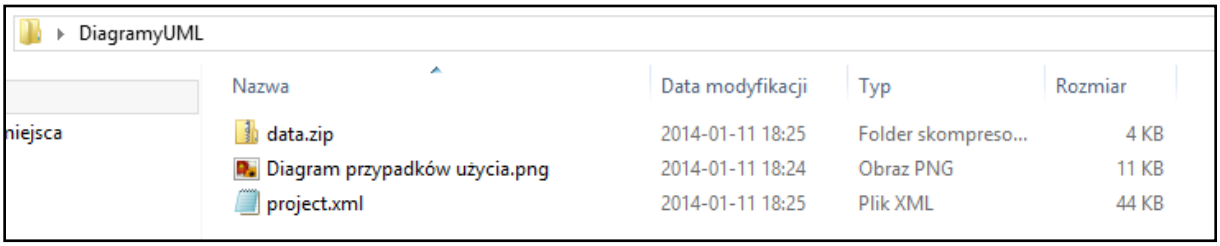
Diagram należy wyeksportować jako plik XML:



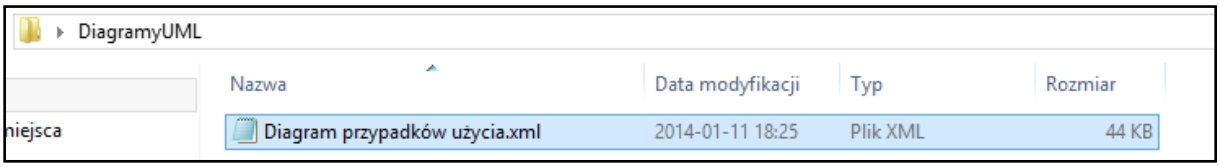
Należy wybrać lokalizację diagramu i odznaczyć opcję "Eksportuj diagramy jako bitmapy":



Wybranej lokalizacji utworzone zostały następujące pliki:



Plik XML naszego diagramu został zapisany pod nazwą project.xml. Pliki data.zip i plik .png są niepotrzebne dlatego usuwamy je z katalogu, a nazwę pliku project.xml zmieniamy na taką, która da nam informację jaki diagram znajduje się w danym pliku. W naszym przykładzie nadaliśmy nazwę "Diagram przypadków użycia":



2. Utworzenie diagramów czynności w Visual Paradigm

Dla każgedo przypadku użycia z diagramu przypadków użycia tworzymy diagram czynności:

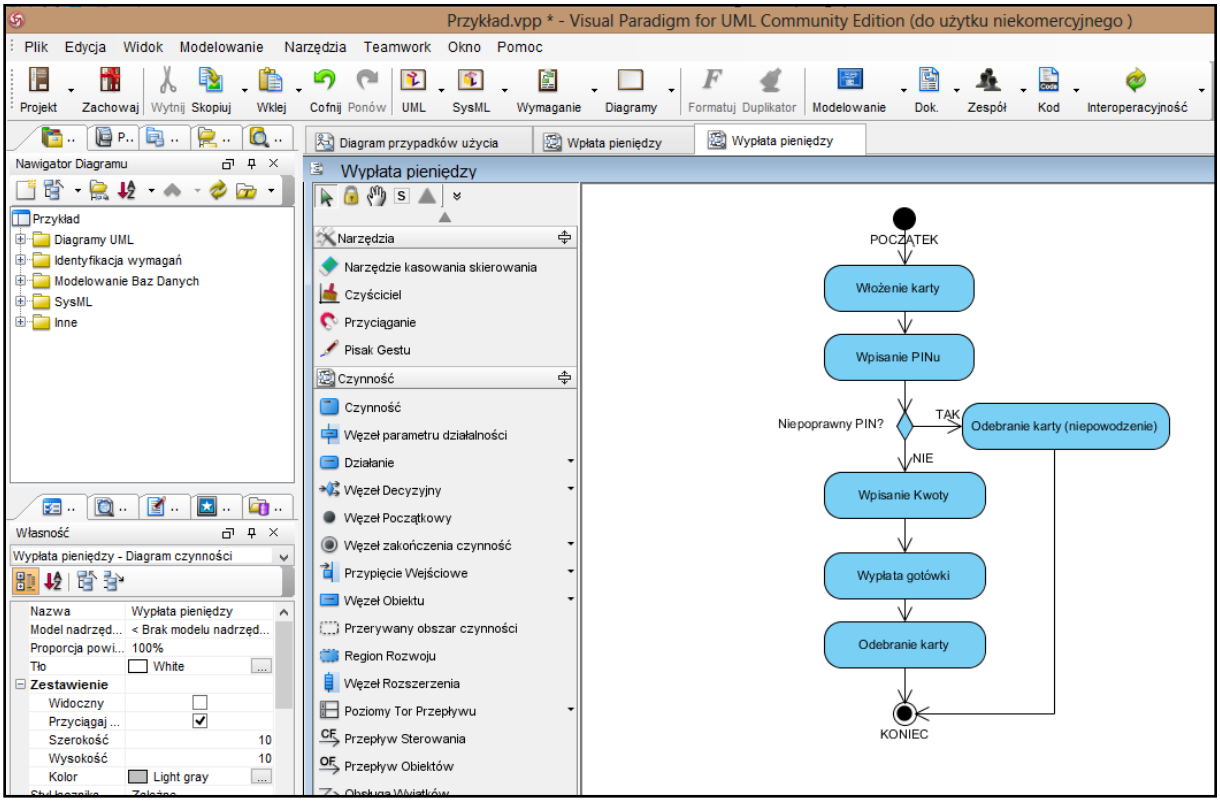
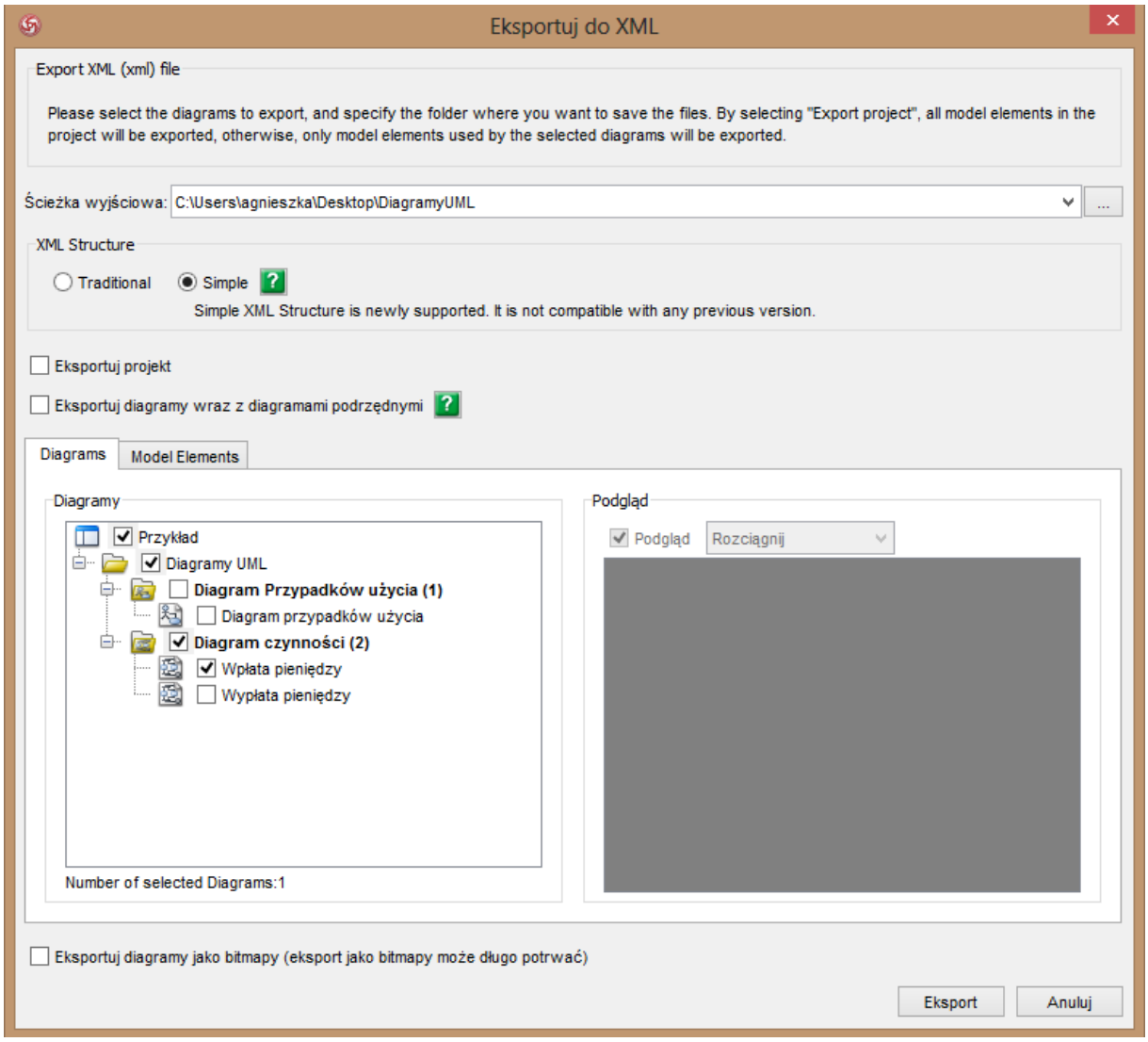


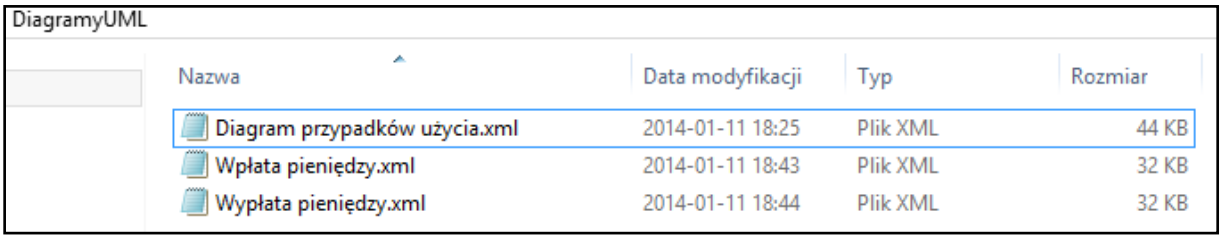
Diagram powinien zawierać wszystkie słowa ze słownika.

Eksportujemy diagram "Wpłata pieniędzy":



Należy zwrócić uwagę, czy zaznaczony jest odpowiedni diagram w sekcji diagramy. wygenerowanemu plikowi project.xml zmieniamy nazwę na "Wpłata pieniędzy.xml". Tak samo generujemy drugi diagram i zmieniamy nazwę na "Wypłata pieniędzy.xml". Niepotrzebne pliku usuwamy.

Ostatecznie otrzymujemy dla naszego przykładu trzy pliki XML: jeden z przypadkami użycia i dwa z diagramami czynności - po jednym dla każdego przypadku użycia:

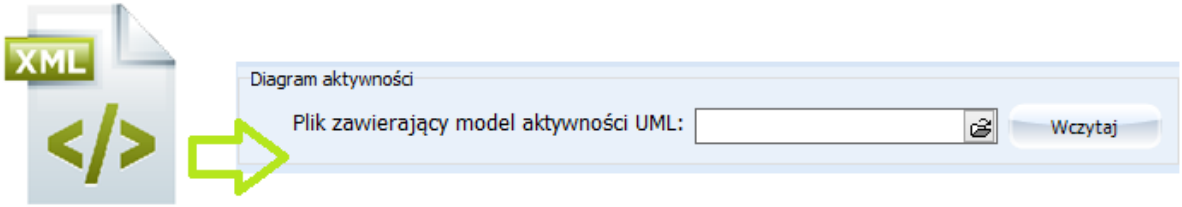


3. Użycie aplikacji Uml2Ltl - wczytanie przypadków użycia

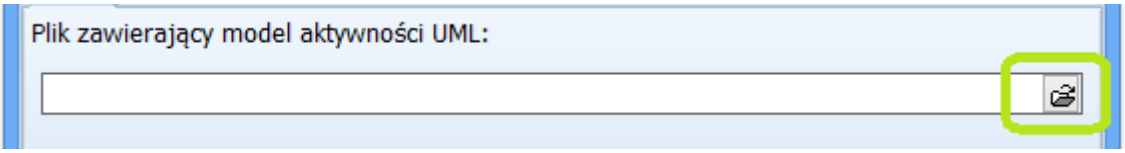
Wczytywanie plików:

1. Metodą drag-and-drop

Aby plik został wczytany wystarczy przeciągnąć go na formę metodą Drag-and-drop w pole w którym znajduje się



lub wybrać jego ścieżkę klikając przycisk z prawej strony pola edycji ścieżki pliku. Ścieżkę można również wpisać ręcznie.

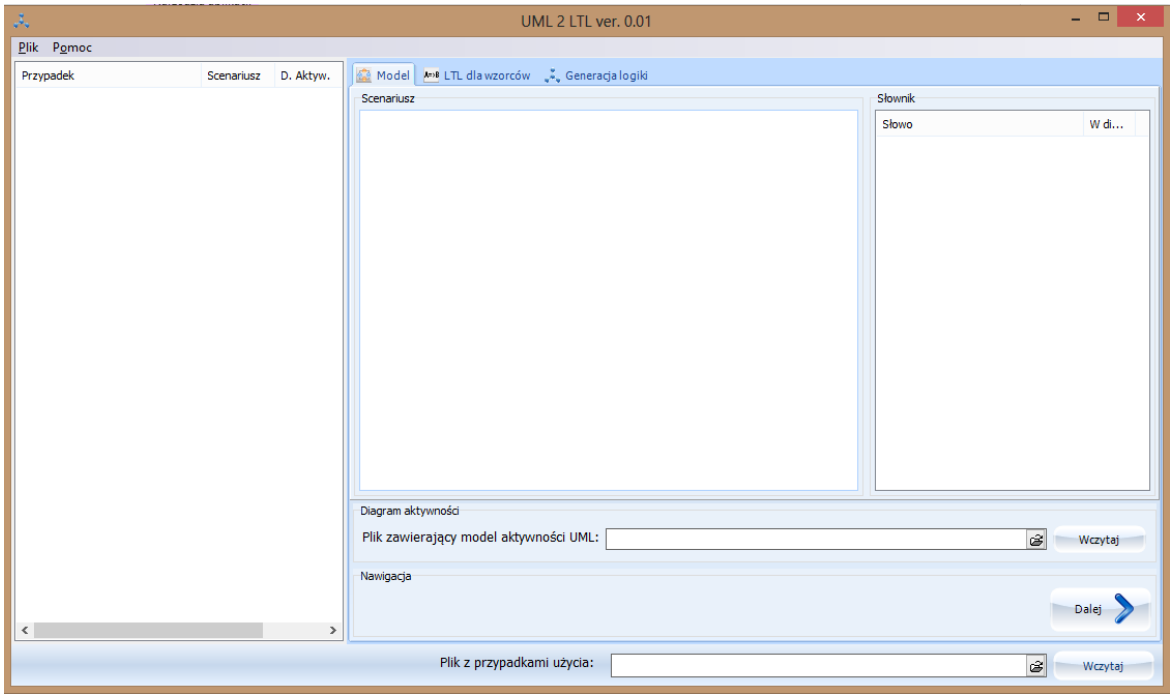


Przykładowe pliki modeli znajdują się w katalogu *res*.

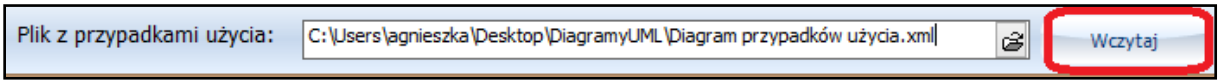
W przypadku pierwszej lub drugiej metody, pole ścieżki uzupełni sie automatycznie. Ścieżkę pliku można również wpisać ręcznie w odpowiedni pole tekstowe.

Przykład użycia aplikacji:

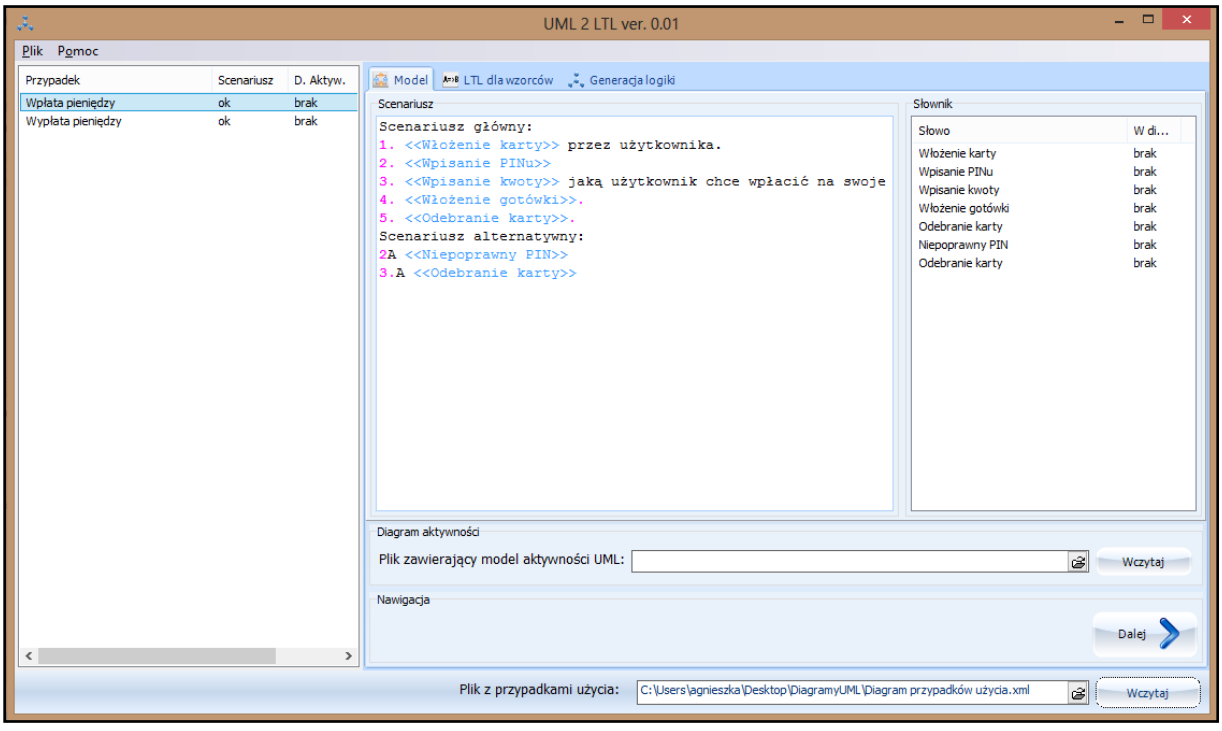
Po odpaleniu przygotowanej aplikacji Uml2Ltl GUI.exe wyświetla się nam następujące okno:



W pierwszej kolejności wczytujemy plik XML z zapisanym diagramem przypadków użycia wybierając odpowiednią ścieżkę pliku oraz zatwierdzając przyciskiem "Wczytaj":



W tym momencie program wczytuje plik przypadków użycia:



Po lewej stronie znajduje się lista przypadków użycia znalezionych w diagramie (w naszym przypadku są to dwa przypadki). Podana jest nazwa przypadku, informacja o tym czy przypadek zawiera scenariusz (ponieważ nasze przypadki posiadały scenariusz, wyświetlane jest "ok", jeśli scenariusza nie ma wyświetlany jest "brak"), oraz informacja o tym czy dodany został diagram aktywności (ponieważ jeszcze nie dodawaliśmy diagramów w programie wyświetlany jest "brak").

Informacje w środkowej części programu. Zakładki Model, LTL dla wzorców oraz Generacja logiki odnoszą się do wyróżnionego przypadku - aby wyróżnić przypadek wystarczy kliknąć na jego nazwę, zostanie on wtedy podświetlony. W zakładce Model w polu Scenariusz wyświetlany jest scenariusz dla przypadku użycia wczytany z pliku XML. Słowa ze słownika podświetlane są na niebiesko oraz wyświetlają się w liście umieszczonej z prawej strony ekranu. Na liście znajduje się również informacja, czy dane słowo znaleziono w diagramie czynności. Ponieważ nie wczytaliśmy jak dotąd diagramów czynności wyświetlony jest "brak".

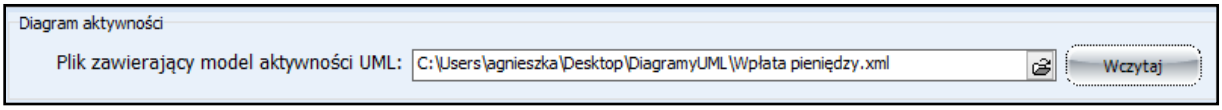
4. Wczytanie diagramów czynności

Diagram czynności dla przypadku użycia wybieramy następująco:

1. Na liście przypadków użycia (z lewej strony ekranu aplikacji) zaznaczamy przypadek użycia, dla którego będziemy wczytywać diagram czynności.

2. W polu "Diagram aktywności" znajdującym się w centralnym miejscu na ekranie znajduje się pole wyboru pliku oznaczone jako "Plik zawierający model aktywności UML". W polu tym wybieramy plik z diagramem czynności.

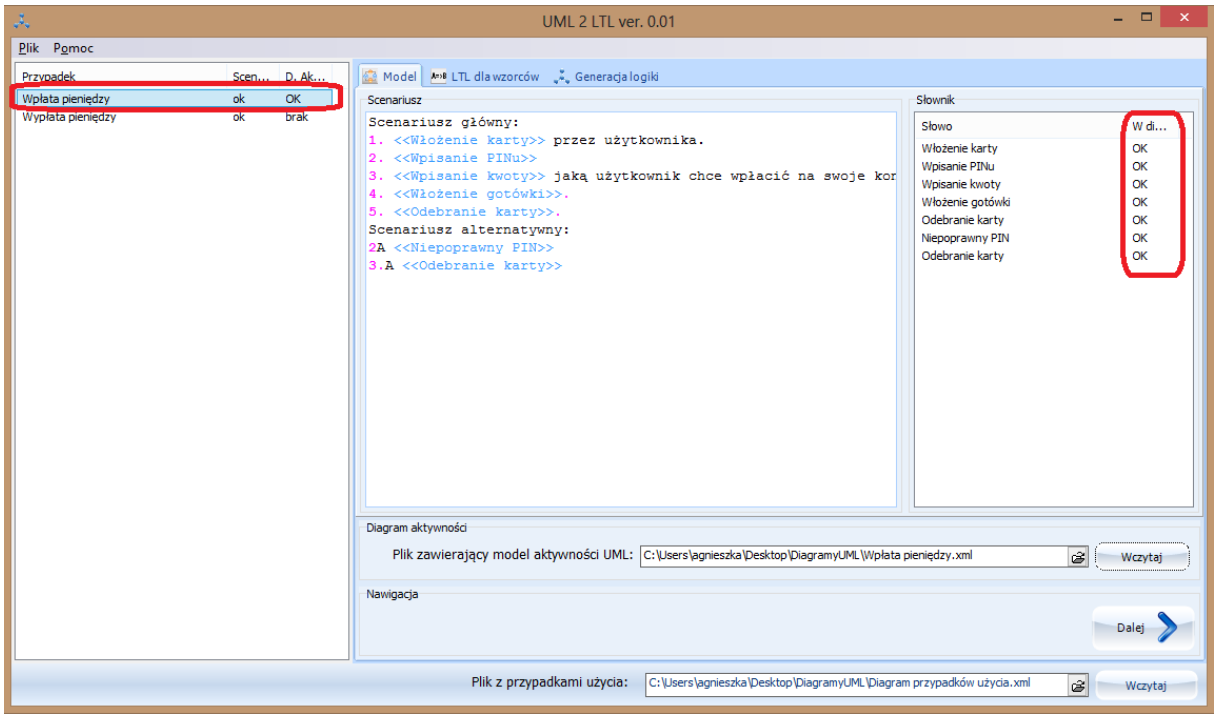
3. Zatwierdzamy przyciskiem "Wczytaj":



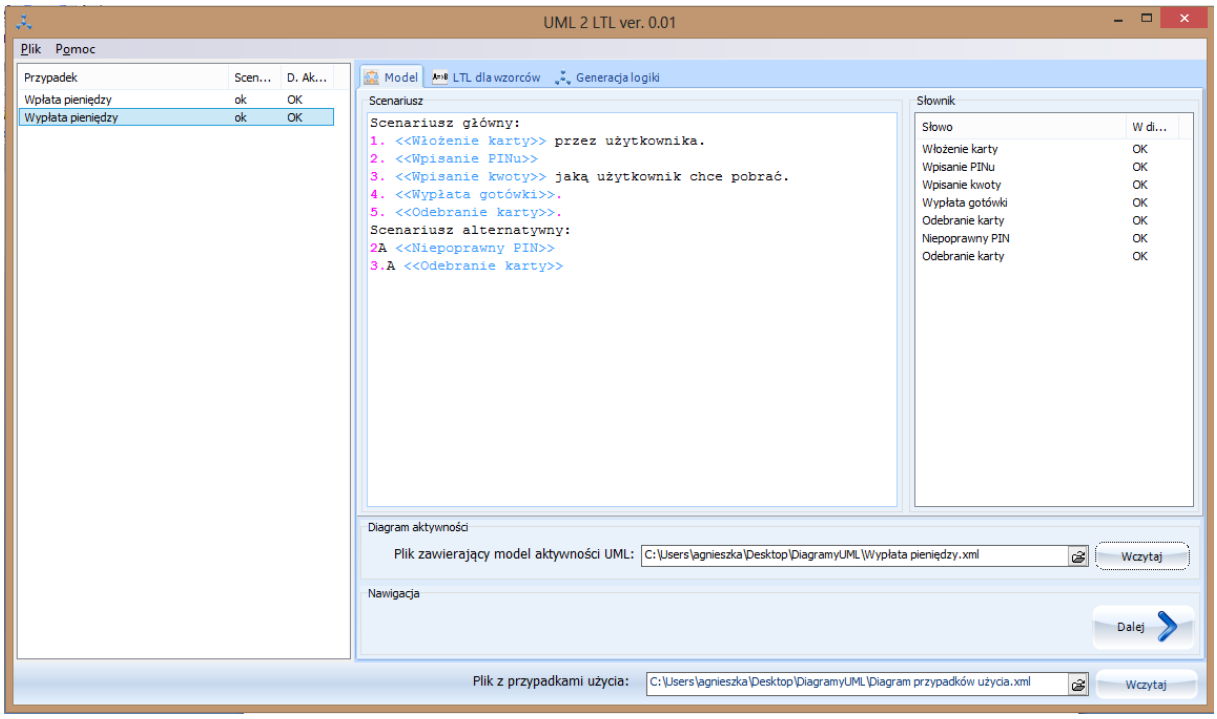
Plik powinien zostać wczytany do programu. Na ekranie widzimy dwie zmiany spowodowane dodaniem przypadku użycia:

1. Na liście przypadków użycia, dla przypadku "Wpłata pieniędzy" pole "D. Aktywności" zmieniło wartość z "brak" na "OK" - oznacza to, że dla danego przypadku użycia wczytaliśmy diagram aktywności.

2. Na liście słów ze słownika, atrybut "W diagramie" zmienił wartość z "brak" na "OK" - oznacza to, że w diagramie czynności dla przypadku testowego odnaleziono dane słowo. Jeśli pole nadal zawierało by wartość "brak" oznaczałoby to, że słowa nie ma w diagramie - ma to być informacja dla użytkownika, że być może musi uzupełnić stworzony wcześniej diagram.



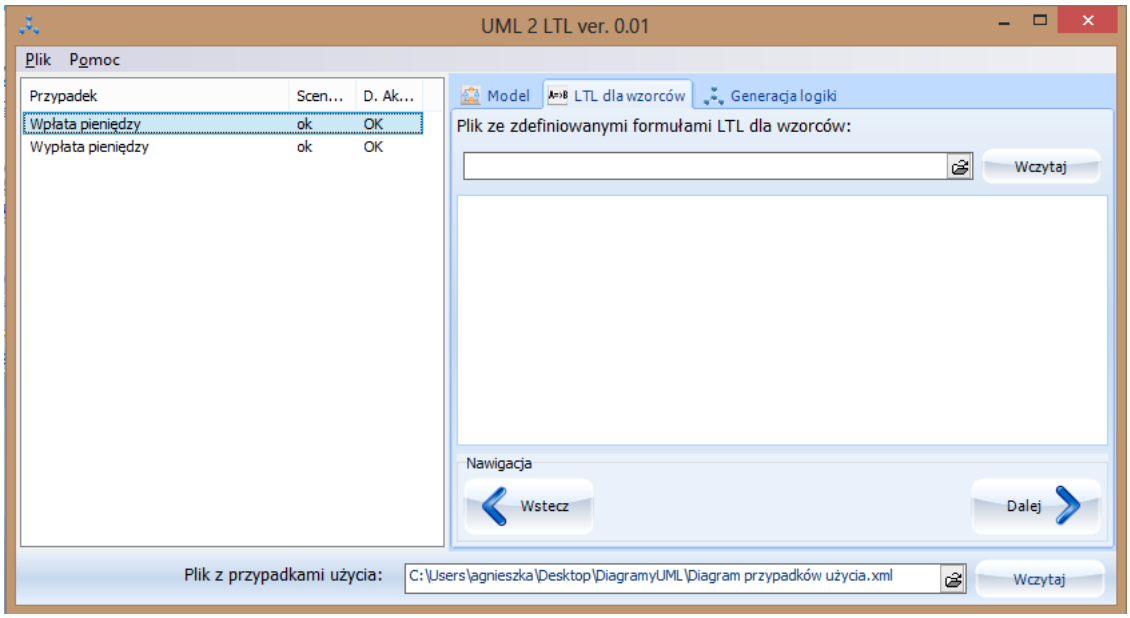
Dla naszego przykładu wczytujemy diagram czynności dla przypadku "Wypłata pieniędzy", rezultat widać na rysunku poniżej:



Klikając przycisk *Dalej* przeniesiemy się do kolejnej zakładki.

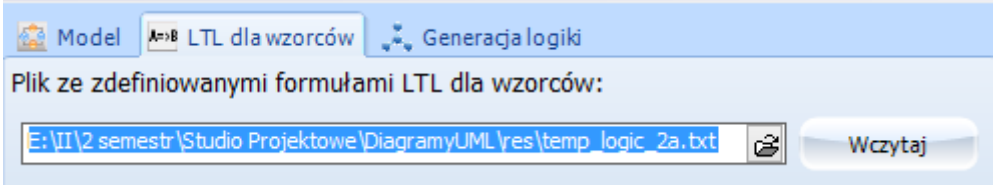
5. Karta LTL dla wzorców

Druga zakładką, jest karta *LTL dla wzorców*.

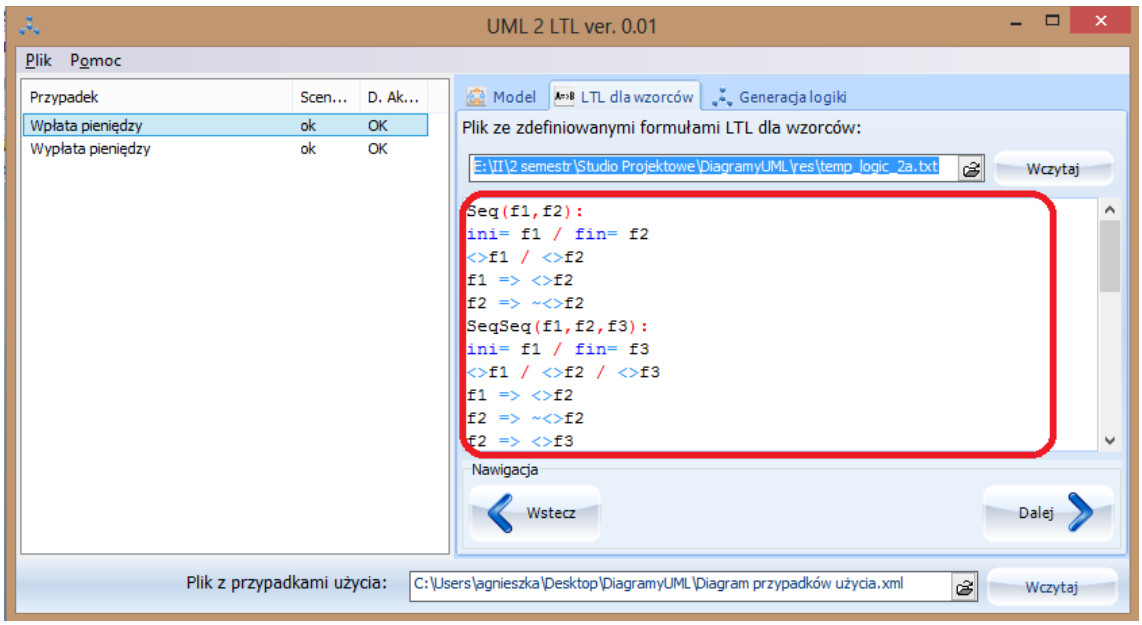


W karcie wczytywany jest plik definicji formuł LTL dla poszczególnych wzorców wyszukiwanych w modelu.

Wczytujemy plik definicji formuł LTL zatwierdzając przyciskiem "Wczytaj":



Plik zostanie automatycznie wczytany na formatkę:



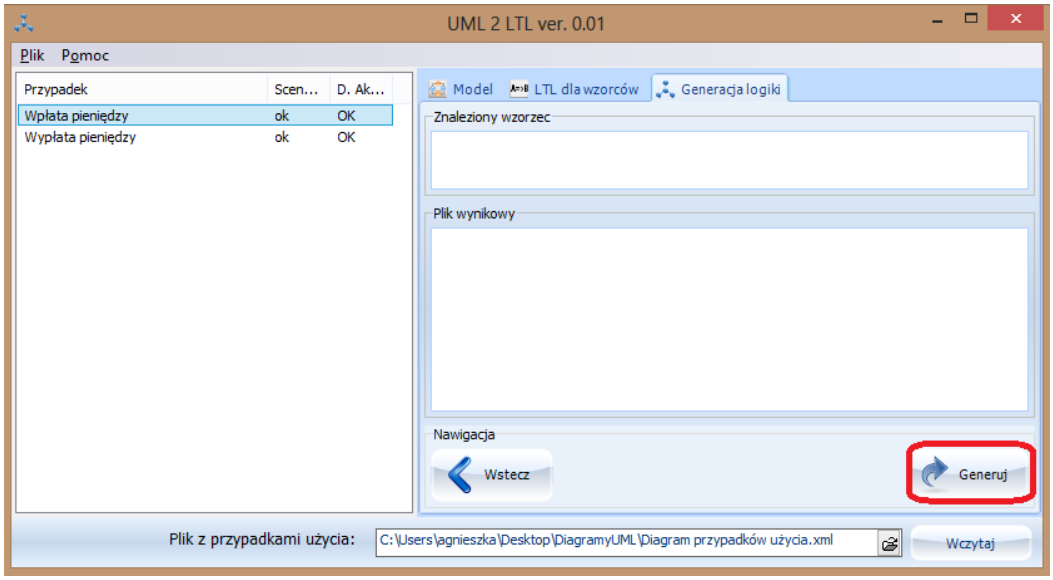
Wczytany plik z regułami LTL wyświetlany jest ze zdefiniowanym specjalnie kolorowaniem składni, ułatwiającym odczyt zawartości.

Przykładowe pliki zawierające logikę LTL dla wzorców znajdują się w katalogu *res*.

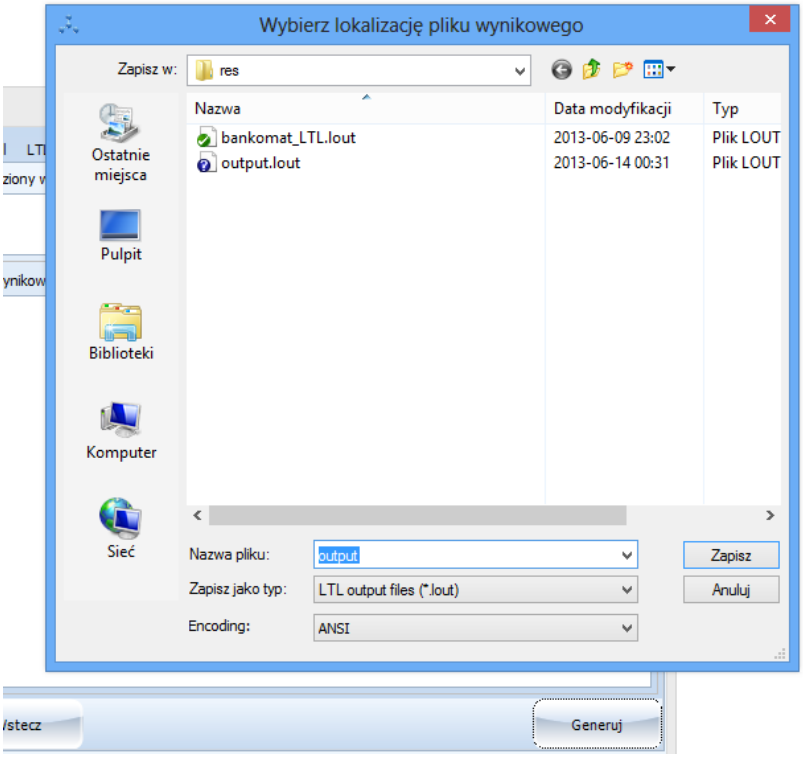
Klikając przycisk *Dalej* przeniesiemy się do kolejnej zakładki.

6. Generacja logiki

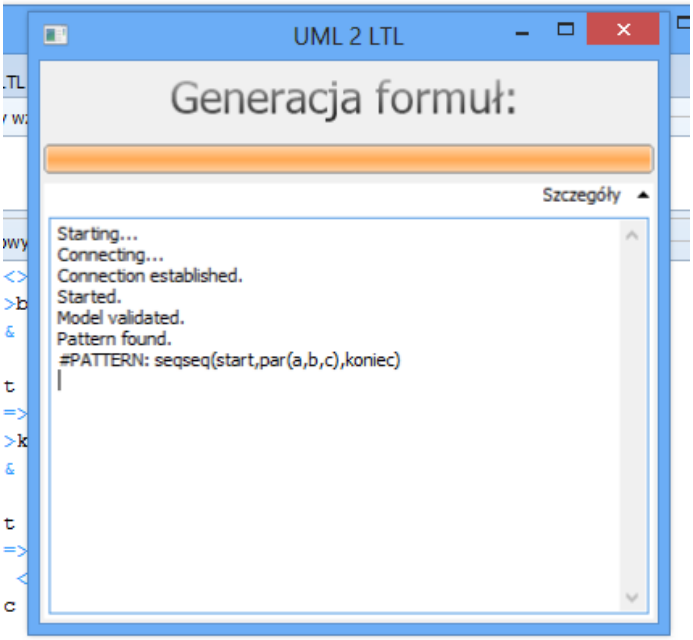
Ostatnią zakładką, jest karta *Generacja logiki*. Aby rozpocząć generację formuł należy wybrać przycisk *Generuj*.



W oknie zapisu pliku należy wybrać ścieżkę do zapisu pliku wynikowego



Następnie wyświetlone zostanie okno wyświetlające postęp przetwarzania modelu i generacji formuł.



Po zakończeniu przetwarzania okno zamknie sie samoistnie. Dane wynikowe wprowadzone zostaną do odpowiednich pól zakładki.