Marek Sawerwain

xText – języki dziedzinowe do zadań specjalnych

DSL to skrót od angielskiego wyrażenia Domain Specific Languages, czyli w wolnym tłumaczeniu wyspecjalizowane/specjalizowane języki dziedzinowe. Realizując różnego typu projekty, często się okazuje, iż warto byłoby utworzyć niewielki program czy skrypt realizujący zadania związane ściśle z projektem, jednakże o składni innej niż typowe uniwersalne języki programowania (zwane też językami GPL – ang. *General Purpose Languages*). Budowa analizatora, a później interpretera czy nawet własnego edytora to niewątpliwie zadanie, które wymaga poświęcenia sporej ilości czasu.

Jednakże utworzenie własnego DSLa nie będzie trudne, jeśli wykorzystamy technologię xText dostępną dla platformy Java. Jest to całościowe rozwiązanie problemu tworzenia własnych języków dziedzinowych, od najprostszych rozwiązań do całkowicie nowych języków programowania.

Możliwości xText najlepiej zareklamować projektem Xtend, czyli pełnym językiem programowania dla maszyny wirtualnej Java, opracowanym w oparciu o technologię xText. Projekt xText jest również zintegrowany ze środowiskiem Eclipse, więc nowe rozwiązania od razu można testować za pomocą możliwości edytora i środowiska Eclipse.

GRAMATYKA NA POCZĄTEK

Większość prezentacji możliwości technologii xText rozpoczyna się od przykładu języka pozdrowień. Dlatego my także rozpoczniemy w ten sposób, choć wprowadzimy także nasze polskie pozdrowienia oprócz angielskiego „Hello".

Naturalnie pierwszą czynnością, jaką trzeba wykonać, jest instalacja środowiska Eclipse oraz pakietu xText. Jak wykonać instalację, szczególnie w kontekście instalacji xText, przedstawiamy w ramce pt. „Instalacja frameworku xText".

Listing 1 przedstawia gramatykę nieskomplikowanego przykładu, a mianowicie języka przywitań. Programy w tym języku pozwalają wpisywać wyrażenia typu:

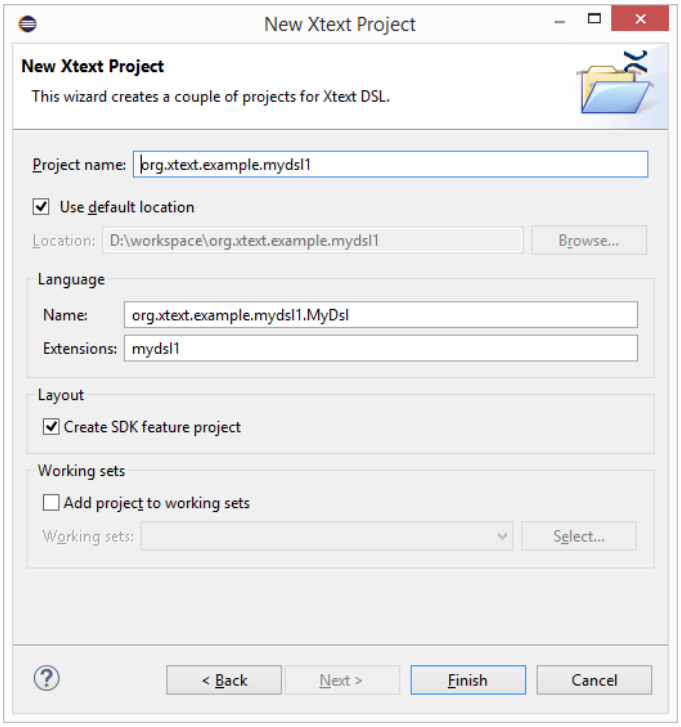
```
Hello <ciąg znaków>!
```

lub też nasze bardziej swojskie i góralskie:

```
Witojcie <ciąg znaków>!
```

Aby zrealizować tego typu język DSL, to po instalacji Eclipsa oraz środowiska xText musimy rozpocząć od utworzenia nowego projektu z menu „File, New, Project…", a następnie w oknie z rodzajami projektów wybieramy projekt typu: „xText Project".

W oknie poprawiamy tylko nazwę projektu oraz pola w sekcji Language, wpisując np. `mydsl` jako nazwę projektowanego języka (należy też zwrócić uwagę na rozszerzenie, jakim będziemy się posługiwać dla pliku źródłowego w danym języku). Utworzone zostaną cztery dodatkowe projekty, ale nas na początek interesuje tylko projekt o nazwie `org.xtext.example.mydsl` (równocześnie jest to nazwa pakietu w Javie). Musimy w nim odszukać plik o nazwie *MyDsl.xtext*, bowiem w nim będziemy tworzyć gramatykę naszego pierwszego DSLa.



Rysunek 1. Tworzenie nowego projektu wykorzystującego technologię xText

Listing 1. Definicja DSL dla języka przywitań

```
grammar org.xtext.example.mydsl.MyDsl with org.eclipse.xtext.
common.Terminals

generate myDsl "http://www.xtext.org/example/mydsl/MyDsl"

Model:
  greetings += ( Greeting | Pozdrowienia )*;

Greeting:
  'Hello' name = POLSTRING '!' ;

Pozdrowienia:
  'Witojcie' name = POLSTRING '!' ;

terminal POLSTRING :
  ( 'a' .. 'z' | 'A' .. 'Z' | 'ą' | 'ć' | 'ę' | 'ł' | 'ó' | 'ś' |
  'ż' | 'ź' | 'Ą' | 'Ć' | 'Ę' | 'Ł' | 'Ó' | 'Ś' | 'Ż' | 'Ź' | '_'
  ) ('a'..'z'|'A'..'Z'|'_'|'0'..'9' | 'ą' | 'ć' | 'ę' | 'ł' | 'ó'
  | 'ś' | 'ż' | 'ź' | 'Ą' | 'Ć' | 'Ę' | 'Ł' | 'Ó' | 'Ś' | 'Ż' |
  'Ź')* ;
```

Po utworzeniu projektu gramatyka będzie już przygotowana, ale nasza wersja została nieco zmieniona względem oryginalnego przykładu, jaki znajduje

się w dokumentacji do pakietu xText i jest proponowany przez Eclipse. Chcemy bowiem posiadać obsługę polskich liter. Domyślnie po słowie `Hello` dopuszczamy ciąg znaków reprezentowany przez regułę o krótkiej nazwie `ID`. Reguła `ID` reprezentuje ciąg znaków dla różnego rodzaju identyfikatorów, np. nazwy zmiennych. Nie są dopuszczalne znaki poza typowymi kodami ASCII. Dlatego musimy utworzyć nową regułę o nazwie `POLSTRING`, gdzie zwyczajnie umieszczamy, jakie znaki mają składać się na identyfikator o nazwie `POLSTRING`.

Regułą główną naszej gramatyki jest `Model`. Zawiera on wyrażenie:

```
greetings += ( Greeting | Pozdrowienia )*;
```

oznaczające, iż język będzie się składał z wyrażeń opisanych przez regułę `Greeting` lub regułę `Pozdrowienia`. Reguł może być dowolnie dużo, ale dopuszczamy także sytuację, gdy nie ma żadnej reguły. Zostało to opisane przez gwiazdkę umieszczoną na końcu wyrażenia. Naturalnie nie ma obowiązku, aby każda gramatyka rozpoczynała się od reguły `Model`.

Same reguły przywitań można zapisać również krócej, bez powoływania oddzielnych reguł:

```
Model:
  greetings += ( Greeting )*;
Greeting:
  'Hello' name = POLSTRING '!' | 'Witojcie' name = POLSTRING '!' ;
```

Wykorzystaliśmy regułę `Greeting`, gdzie jako alternatywa został podany drugi wariant przywitania.

Pozostało pytanie, jak uruchomić tę gramatykę, i sprawdzić możliwość edycji „programu" w naszym języku za pomocą środowiska Eclipse. W pierwszej kolejności trzeba wybrać np. z menu podręcznego opcję „Generate xText Artifacts". Menu podręczne wywołujemy, jeśli w edytorze mamy otwarty plik *MyDsl.xtext,* i wymienioną opcję znajdziemy z „Run As…". Za pierwszym razem zostaniemy zapytani o zgodę na instalację pakietu ANTLR 3, który jest generatorem parserów. Pytanie to pojawia się w konsoli. Po krótkiej chwili gramatyka oraz reszta projektu xText zostanie przebudowana.

Uruchomienie edytora Eclipse obsługującego nasz nowy DSL sprowadza się do wybrania opcji „Eclipse Application" z menu „Run As…". Po uruchomieniu Eclipsa, należy założyć podstawowy ogólny projekt, do którego musimy dodać jeden plik o rozszerzeniu zgodnym z naszym językiem przywitań, czyli „mydsl". Pojawi się pytanie, czy chcemy dołączyć xText do projektu, na co naturalnie powinniśmy się zgodzić. W ten sposób edytor Eclipse będzie poprawnie formatował kod źródłowy naszego języka przywitań.

Instalacja frameworku xText

Trudno pisać o instalacji środowiska Eclipse, bowiem np. w przypadku systemów Linux dostępne są gotowe pakiety, z których instalujemy Eclipsa. W przypadku Windows, ze strony domowej projektu Eclipse, możemy ściągnąć gotowe wersje instalacyjne. Instalować powinniśmy wersję co najmniej Luna lub Mars. Przy czym w przypadku całkowicie samodzielnej instalacji trzeba sprawdzić, czy mamy dostępną Javę.

Dla osób, które chcą łatwo rozpocząć pracę z xText, najwygodniej będzie polecić ściągnięcie ze strony xText wersję Eclipsa, która posiada już zainstalowany pakiet xText. Dostępne są wersje dla systemów Linux, Windows oraz Mac OS.

Instalacja xText w środowisku Eclipse również nie jest skomplikowana. Dokonujemy jej poprzez wybór z menu „Help" opcji „Install New Software…", gdzie poprzez przycisk „Add…" dodajemy nowe źródło pakietu reprezentującego xText. Adres znajdziemy na stronie xText i jest on następujący:

http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/

Reprezentuje on wersję stabilną. Jeśli będziemy instalować samodzielnie, to warto podczas instalacji wskazać tylko dwa projekty o nazwach dokładnie Xtend oraz Xtext (w artykule stosujemy inne nazwy z małą literą „x" na początku).

JESZCZE O GRAMATYCE

Proces tworzenia gramatyki to podstawowe zagadnienie w całym zadaniu kreacji nowego wyspecjalizowanego języka programowania. Plik z gramatyką rozpoczyna się od słowa `grammar`:

```
grammar ID with ID
```

Możemy określić nazwę pakietu, który reprezentuje gramatykę, a także po słowie `with` wskazać pakiet, z którego będziemy importować dodatkowe definicje ułatwiające tworzenie gramatyki. Podobną rolę pełni słowo `import`, po którym określamy, jakie elementy dołączamy do naszej gramatyki. Można nawet powiedzieć, że gramatyka w xText może niejako dziedziczyć z innej gramatyki. Korzystamy z tej możliwości w pierwszym oraz w drugim przykładzie, dołączamy bowiem gotowe definicje wyrażeń czy też symboli terminalnych.

Nazwę naszej gramatyki określamy poprzez słowo kluczowe `generate`. Po tych wstępnych definicjach podajemy reguły opisujące DSLa. Pierwsza reguła oznacza też regułę główną. Składnia jest następująca:

```
nazwa_reguły:
  definicje lub wywołania innych reguł
;
```

Nasza podstawowa reguła odnosząca się do języka przywitań może przedstawiać się następująco:

```
Model:
  greetings += ( Greeting )*;
```

Oznacza to, że język przywitań składa się ze zbioru przywitań, gdzie zakłada się, iż liczba przywitań to zero lub więcej (oznaczono to znakiem gwiazdki). Pojedyncze przywitanie jest natomiast opisane przez regułę `Greeting`.

Dodatkowe oznaczenia to konsekwencja faktu, iż pakiet xText stosuje tzw. rozszerzoną notację Backusa-Naura. Pozwala ona stosować dodatkowe oznaczenia, czy dana reguła ma występować np. wiele razy, czy też przynajmniej raz. Pozwala to na uproszczenie wielu zapisów, bowiem nie trzeba tworzyć rekurencyjnych konstrukcji opisujących krotność stosowanych reguł.

Dodatkowych oznaczeń naturalnie jest więcej. Dopuszcza się stosowanie znaku plus, oznaczającego, że dana reguła jest stosowana przynajmniej raz lub więcej razy. Znak zapytania oznacza, że reguła może zostać użyta tylko raz bądź może być pominięta. Natomiast brak dodatkowych symboli oznacza, iż dana reguła ma zostać zastosowana dokładnie raz.

Inna przykładowa reguła, pochodząca z dokumentacji, ma postać:

```
State: 'state' name=ID
  ('actions' '{' (actions+=[Command])+ '}')?
  (transitions+=Transition)*
'end';
```

Opisuje ona stan z maszyny stanów, jaką projektuje się za pomocą dedykowanego DSLa (przykład pochodzi z dokumentacji do xText). Reguła ustala następującą składnię: na początku bloku występuje słowo kluczowe `state`, a po nim nazwa stanu. Następnie opcjonalnie (na co wskazuje znak zapytania na końcu reguły) mogą wystąpić nazwy akcji, które są oddzielone spacjami. Wprowadzenie przecinka nieco utrudniłoby zapis gramatyki. Drugą częścią opisu stanu są tranzycje, także są opcjonalne, dopuszcza się wystąpienie wielu tranzycji bądź żadnej. Przykładowy zapis stanu, jaki spełnia podaną wcześniej regułę, jest następujący:

```
state unlockedPanel
  actions {unlockPanel lockDoor}
  panelClosed => idle
end
```

W gramatykach ważną rolę pełnią reguły opisujące tzw. symbole terminalne. Zazwyczaj są to różnego rodzaju identyfikatory, nazwy zmiennych, a także