26. Języki dziedzinowe – idea, narzędzia wspierające

## Języki dziedzinowe (DSL – Domain Specific Language)

* To języki programowania:
  + charakteryzujące się ograniczoną ekspresywnością,
  + przeznaczone do specyficznych zadań
  + skupione na wąskiej, szczegółowej dziedzinie
  + zaprojektowane w celu rozwiązania określonej dziedziny problemów
* Są przystosowane do rozwiązywania określonej dziedziny problemów, określonej reprezentacji problemów lub określonej techniki rozwiązywania problemów
* Języki specyficzne dla domeny/biznesu, które pozwalają na dokładniejszy opis dziedziny której dotyczą. Wyzwania:
  + Podniesienie poziomu abstrakcji dziedziny
  + Próba formalnego opisu problemu i rozwiązania

## Cele DSL

* Mniejsze skomplikowanie
* Lepszy opis dziedziny
* Zminimalizowana „nadmiarowości”

## Języki dziedzinowe a modelowanie i wytwarzanie oprogramowania

* W wytwarzaniu oprogramowania stanowią przeciwieństwo języków programowania ogólnego przeznaczenia (C, Java) oraz języków modelowania ogólnego przeznaczenia (UML)
* Języki dziedzinowe mogą być wykorzystywane do reprezentacji modeli danych. Dla każdego z modeli – konceptualnego, logicznego i fizycznego – możliwe jest zdefiniowanie odrębnych języków dziedzinowych.
* Języki dziedzinowe są pośrednio związane z zadaniem walidacji modeli, w szczególności modelu konceptualnego. Model konceptualny jest podstawowym, zwykle pierwszym, odzwierciedleniem modelowanego wycinka rzeczywistości. Adekwatność tego modelu jest warunkiem koniecznym poprawności całego procesu modelowania. Walidacja modelu konceptualnego polega na jego przeglądzie i analizie przez eksperta dziedzinowego. Czytelność modelu, a zwłaszcza to, czy model ten operuje pojęciami bezpośrednio związanymi z dziedziną zastosowania, decydują o trudności i poprawności procesu walidacji. Możliwość doboru odpowiedniego języka dziedzinowego wpływa na proces walidacji. W wielu zastosowaniach może okazać się, że takie języki są z tego względu wygodniejsze niż ogólne języki modelowania jak UML, czy BPML.
* Transformacja modeli wiąże się pośrednio z językami dziedzinowymi. W procesie wytwarzania oprogramowania modele są kolejno przekształcane – model reprezentowany przez jeden język dziedzinowy może być przetransformowany w kolejny reprezentowany w innym języku dziedzinowym. Na przykład, środowisko MS Visual Studio umożliwia reprezentację definiowanych transformacji w językach C# oraz Visual Basic. Środowisko udostępnia ułatwienia pozwalające na operowanie w języku programowania elementami transformowanych modeli. Takie podejście do definiowania transformacji może być w wielu przypadkach bardziej efektywne niż stosowanie języków specjalnie przeznaczonych do definiowania transformacji, na przykład QVT .

## Klaryfikacja języków dziedzinowych

1. Ze względu na stosunek do języka bazowego (języka na podstawie którego zostały utworzone)
   1. **Zewnętrzne (ex - DSL)**
      * Napisane w innym języku niż język bazowy
      * Używane z wykorzystaniem interpretera
      * Np. Samodzielne języki (wyrażenia regularne), pliki konfiguracyjne XML
      * Zalety:
        + Dowolność składni (łatwiejsze powiązanie z dziedziną
      * Wady:
        + Konieczność stworzenia interpretera
        + Brak środowiska (IDE, debugger)
        + Problemy przy komunikacji z językiem bazowym (oddzielne symbole)
   2. **Wewnętrzne (in – DSL)**
      * Składnia taka sama jak języka bazowego, ograniczone możliwości
      * Np. LISP
      * Zalety:
        + Dostępność środowisk (IDE, debugger)
        + Brak problemów komunikacyjnych z językiem bazowym
        + Interpreter nie jest konieczny
      * Wady:
        + Skomplikowane dla nie-programistów
        + Ograniczone możliwości dostosowania do dziedziny
2. Ze względu na reprezentacje
   1. Graficzne – używają do opisu symboli graficznych zamiast opisu tekstowego
   2. Tekstowe – najbardziej powszechne, opisują system za pomocą określonej, charakterystycznej dla nich składni.

## Przykłady języków dziedzinowych

* **SQL dla zapytań do relacyjnych baz danych.**
* SQL to strukturalny język zapytań używany do tworzenia, modyfikowania struktury baz danych oraz do umieszczania i pobierania danych z baz danych. Można go uznać za język dziedzinowy, gdyż jest on specyficzny dla określonej dziedziny (w przypadku dostępu i zarządzania relacyjną bazą danych) oraz może być wywoływany z innej aplikacji.
* Z racji tego, że SQL ma o wiele więcej funkcji i słów kluczowych niż inne języki skryptowe, jest on uważany za język sam w sobie.
* **HTML**
* Język dziedzinowy zdefiniowany na podstawie SGML - *aplikacja SGML* (stał się najbardziej popularnym zastosowaniem SGML).
* **SGML** (Standard Generalized Markup Languege) - międzynarodowa norma (ISO 8879), dotycząca strukturalizacji dokumentów elektronicznych. Nie jest to język, ale sposób formalny definiowania języków znakowania, metajęzyk.
* **XSLT, XMI, SVG**
* Języki dziedzinowe oparte na XML (*aplikacje XML*).
* **XML** (Extensible Markup Language) to udoskonalony, znacząco uproszczony następca SGML, zaprojektowany na potrzeby wymiany danych w środowisku internetowym. Podobnie jak SGML - stanowi metajęzyk, toteż nie definiuje - w przeciwieństwie do HTML - znaczników, a jedynie konwencję składniową.
* **XSLT** – oparty na XML’u język przekształcający dokumenty XML na tekst, HTML lub XHTML (dokumenty zgodne ze składną XML)
* **Wyrażenia regularne**
* To wzorce, które opisują łańcuchy symboli.
* Wyrażenia regularne mogą określać zbiór pasujących łańcuchów, mogą również wyszczególniać istotne części łańcucha.
* Wyrażenia regularne to w informatyce teoretycznej ciągi znaków pozwalające opisywać języki regularne. W praktyce znalazły bardzo szerokie zastosowanie, pozwalają bowiem w łatwy sposób opisywać wzorce tekstu, natomiast istniejące algorytmy w efektywny sposób określają, czy podany ciąg znaków pasuje do wzorca lub wyszukują w tekście wystąpienia wzorca.
* **Języki wyrażeń i makr w arkuszach kalkulacyjnych** 
  + Makro – zestaw rozkazów realizujący algorytm komputerowy przeznaczony do wykonania przez jakąś aplikacje (edytor, arkusz kalkulacyjny, program graficzny). Stosowane zwykle w celu automatyzacji pewnych czynności lub dokonania zmian w dokumentach bez interakcji z użytkownikiem. Makra pisane są zwykle w skryptowych językach programowania, wykonywanych przez interpreter wbudowany w aplikacje, w których są uruchamiane.

## Proces definiowania DSL

1. **Definicja meta-modelu języka DSL** czyli abstrakcyjnej składni języka (schematu dla abstrakcyjnej reprezentacji) . Najważniejszy etap procesu tworzenia języka dziedzinowego. Do budowy meta-modelu używa się specjalnego języka dziedzinowego, posiadającego własną notację graficzną. Język pełni analogiczną rolę jak MOF dla języka UML, definiując meta-metapoziom dla języków DSL.
2. **\*Definicja ograniczeń składniowych oraz reguł do ich sprawdzania** – reguły walidacji modelu (np. nazwy klas unikalne).
3. **Definicja składni konkretnej (tekstowej lub graficznej)** - elementom języka DSL przypisywana jest notacja graficzna lub tekstowa
4. **Generacja edytora modeli** (modele tekstowe lub graficzne). Dla zdefiniowanej reprezentacji graficznej lub tekstowej języka automatycznie generowany jest edytor modelu. Edytor ma pozwolić ludziom na manipulację abstrakcyjną reprezentacją.
5. **Definicja transformacji (generatora definiującego semantykę DSL)** – opisuje jak przetłumaczyć abstrakcyjną reprezentację na reprezentacje wykonywalną. Implementacja transformacji służących generowaniu artefaktów z modeli. Artefakty wynikowe mogą być plikami tekstowymi (kod źródłowy, plik konfiguracyjny, skrypt języka SQL, dokumentacja).

\*(etap opcjonalny, reszta obligatoryjna)

Fabryki języków (Language Workbench)

* **Narzędzia wspierające tworzenie języków dziedzinowych**.
* Pozwalają na definicję DSL za pomocą 3 głównych elementów: schematu (definicji języka), edytora i generatora.
* Funkcje związane z definicją języka:
  + Definicja abstrakcyjnej składni języka (meta-modelu)
  + Wsparcie dla wybranego meta-meta modelu języka (MOF, Ecore, Kermeta, UML)
  + Definicja ograniczeo składniowych, np. nazwy klas unikalne
* Funkcje związane z edytorem modeli:
  + Definicja składni konkretnej (tekstowej lub graficznej)
  + Wygenerowanie edytora modeli
* Funkcje związane z generacją kodu z modeli:
  + Definicja transformacji M2T (model-to-text)

## Przykłady narzędzi

|  |  |  |
| --- | --- | --- |
| **Etapy procesu definiowania DSL** | **oAW**  **openArchitectureWare** | **VSVaMSDK**  **Visual Studio Visualization and Modeling SDK** |
| Definicja abstrakcyjnej składni języka (meta-modelu) | Np. za pomocą Eclipse Modeling Framework (EMF) lub narzędzia CASE  Meta-model języka: Ecore | Własne edytory.  Meta-model: własny. Postać: klasy dziedzinowe + relacje zawierania i referencji |
| Definicja ograniczeń składniowych | Język Check | Reguły walidacyjne C# lub Visual Basic |
| Definicja składni konkretnej | Język XText | Możliwość dostosowania edytora graficznego modeli – pasek narządzi, wygląd: kształty, obrazki, dekoratory, konektory, porty, |
| Generacja edytora modeli | Modele tekstowe | Modele graficzne |
| Definicja transformacji M2T (generator adefiniującego semantykę DSL) | Język XPand + pomocniczy język XTend (wyrażenia nawigacyjne) | Język T4 (Text Template Transformation Toolkit) |

## Przykład tworzenia DSL w środowisku Visual Studio

(źródło: Języki dziedzinowe w modelowaniu danych w kontekście MDA, Iwona Dubielewicz, Bogumiła Hnatkowska, Zbigniew Huzar, Lech Tuzinkiewicz)

* **Visual Studio Visualization and Modeling SDK** to zestaw narzędzi umożliwiających tworzenie graficznych języków dziedzinowych i automatyczne tworzenie plików wynikowych z modeli będących instancjami tych języków.
* Tworzenie języka dziedzinowego przebiega w pięciu podstawowych etapach:
  1. **Stworzenie projektu w środowisku Visual Studio.**

W ramach etapu 1 podejmowane są decyzje dotyczące m.in. nazwy języka dziedzinowe-go oraz rozszerzenia nazwy pliku interpretowanego przez środowisko jako model języka dziedzinowego. Nowobudowany język może wykorzystywać istniejące szablony projektów języków, między innymi szablon oparty o diagramy klas.

* 1. **Definicja meta-modelu języka DSL (składnia abstrakcyjna)**

Najważniejszym etapem procesu tworzenia języka dziedzinowego jest etap 2 definiowa-nia meta-modelu języka, czyli składni abstrakcyjnej. Do budowy meta-modelu używa się specjalnego języka dziedzinowego, posiadającego własną notację graficzną. Język pełni analogiczną rolę jak MOF dla języka UML, definiując meta-metapoziom dla języków DSL. Wprowadza on dwa podstawowe pojęcia: klasy i relacji łączących klasy. Klasom i relacjom można przypisać własności. Wśród relacji wyróżnia się:

* powiązanie (*reference*), reprezentowane przerywaną linią – odpowiednik asocjacji UML,
* zawieranie (*embedding*), reprezentowane ciągłą linią – odpowiednik kompozycji z UML,

Końcom relacji powiązania i zawierania przypisane są informacja o obligatoryjności/opcjonalności związku oraz o licznościach końców: 1 – dokładnie jeden; 0 – zero lub jeden; \* – wiele; + – co najmniej jeden.

Dla tak opracowanego meta-modelu generowany jest kod języka C# (lub Visual Basic): dla każdego elementu meta-modelu (klasy, relacji) generowana jest klasa o nazwie identycznej z nazwą elementu.

* 1. **Definicja składni konkretnej**

W ramach etapu 3 definiowana jest dla języka DSL konkretna składnia graficzna. Elementom języka DSL przypisywana jest notacja graficzna za pomocą kolejnego języka dziedzinowego. Autor języka DSL wskazuje geometryczne kształty, które będą używane do tworzenia instancji modeli DSL. Na przykład, do reprezentacji klas wybierane są prostokąty, a relacje pomiędzy elementami prezentowane są za pomocą linii o określonym typie (ciągła, przerywana, itp.) oraz zakończeniu (strzałka, romb, itp.).

Dla zdefiniowanej reprezentacji graficznej języka automatycznie generowany jest kod w języku C# lub Visual Basic, który jest jego edytorem graficznym. Dodatkowo jest generowany zestaw widoków dla modelu reprezentowanego w zdefiniowanym języku dziedzinowym.

* 1. **Definicja ograniczeń (reguły walidacji modelu)**

W ramach etapu 4 można zdefiniować dodatkowe ograniczenia składniowe (składnię kontekstowa) modelu oraz opracować reguły ich sprawdzania. Obliczanie ograniczeń składni kontekstowej może być inicjowane pewnymi zdarzeniami (na przykład na życzenie użytkownika, przy zapisie/odczycie modelu).

* 1. **Definicja transformacji**

Ostatnim etapem tworzenia języka dziedzinowego jest implementacja transformacji służących generowaniu artefaktów z modeli. Artefakty wynikowe mogą być plikami tekstowymi (kod źródłowy, plik konfiguracyjny, skrypt języka SQL, dokumentacja). Sposób generacji jest oparty o mechanizm szablonów wyrażanych w kolejnym języku dziedzinowym. Szablon jest plikiem zawierającym, oprócz fragmentów tekstu, które pojawią się na wyjściu bez zmian, specjalne znaczniki, które w trakcie transformacji zostaną zastąpione danymi z transformowanego modelu lub spowodują uruchomienie pewnego działania. Znaczniki te mogą być dyrektywami (konfiguracja procesu transformacji), instrukcjami, czy wyrażenia-mi. Każdemu rodzajowi artefaktu odpowiada jeden szablon.

W przypadku, gdy artefaktem wynikowym ma być model wyrażony w innym języku DSL, potrzebne jest stworzenie programu, który dokonuje stosownego odwzorowania.

* Etap 4 opracowania reguł walidacji modelu jest opcjonalny. Wszystkie pozostałe są obligatoryjne.